8-2006

Automated Design Space Exploration for Digital Hardware

Jonathan Logan Turnmire
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Jonathan Logan Turnmire entitled "Automated Design Space Exploration for Digital Hardware." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald Bouldin, Major Professor

We have read this thesis and recommend its acceptance:

Syed Islam, Gregory Peterson

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Jonathan Logan Turnmire entitled "Automated Design Space Exploration for Digital Hardware." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Donald Bouldin
Major Professor

We have read this thesis
and recommend its acceptance:

Syed Islam

Gregory Peterson

Accepted for the Council:

Anne Mayhew
Vice Chancellor and
Dean of Graduate Studies

(Original signatures are on file with official student records.)

AUTOMATED DESIGN SPACE EXPLORATION FOR DIGITAL HARDWARE

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Jonathan Logan Turnmire
August 2006

Abstract

Hardware design in VHDL can be a tedious and daunting task. One of the final steps is to realize the design in hardware. Whether targeting a field programmable gate array (FPGA) or an application specific integrated circuit (ASIC), some time needs to be spent optimizing the design to meet project requirements. This paper details the use of the Synplicity physical synthesis toolsets for FPGAs on the UT Solaris workstation cluster, evaluation of the Synplicity physical synthesis toolsets to determine their efficacy in achieving multiple solutions over the power-delay-area design space and evaluation of the Liberator toolset for ASIC design to determine its efficacy in achieving multiple solutions over the power-delay-area design space.

Walkthrough tutorials were created to assist users in becoming familiar with using Synplify and Amplify. An automation script was run with Synplify Pro on a VHDL implementation of AES targeting a Xilinx Virtex-II Pro (XC2VP30-6). Synplify generated solutions varying from 74.7MHz to 112.8MHz with an area utilization of 44% to 45%. Liberator invoked the design space search algorithm (DSSA) developed by Fuat Karakaya against each of five circuits. The DSSA was stopped after generating 32 points for the 32-bit adder, 32-bit multiplier, 32-bit complex multiplier and the 32-bit FIR filter. The 64-bit FFT stopped after 14 days of run time. The power-delay-area product (PDAP) ranged from 1.85 to 4.3 $\mu J*\mu m^2$ for the adder, 1.9 to 3.4 $\mu J*\mu m^2$ for the multiplier, 39.6 to 90.5 $\mu J*\mu m^2$ for the complex multiplier, 23 to 33.6 $\mu J*\mu m^2$ for the FIR and 91.8 to 133.7 $\mu J*\mu m^2$ for the FFT.

Table of Contents

List of Figures

Chapter 1 Introduction

In the course of developing digital hardware, many different approaches may be used to obtain solutions. Some of the advantages of having multiple methods are the results may be different in power usage, calculation delay or area consumed. These varying solutions can be of great use in different applications. Low power usage is needed for portable or battery operated electronics. Minimized area is useful for minimizing cost and maximizing yield. Minimized delay is applicable in applications where speed is the necessitating factor.

1.2 Goals and Expected Contributions

This thesis utilizes electronic design automation tools to produce both FPGA and ASIC implementations over a range of power, area and delay solutions. The FPGA path uses Synplicity's Synplify synthesis and Amplify floorplanning software to target a Xilinx Virtex-II Pro (XC2VP30-6). The ASIC path uses Synopsys's DesignCompiler and PowerCompiler to target the 180-nanometer CMOS process offered by the Taiwan Semiconductor Manufacturing Company (TSMC).

Expected contributions include a description of how to use the Synplicity physical synthesis toolsets for FPGAs on the UT Solaris workstation cluster, evaluation of the Synplicity physical synthesis toolsets to determine their efficacy in achieving multiple solutions over the power-delay-area design space and evaluation of the Liberator toolset to determine its efficacy in achieving multiple solutions over the power-delay-area design space.

1.3 Outline

The second chapter of this paper deals with the needs for varying solutions. The third chapter gives an overview of the software used. The fourth section covers the results obtained for various modules. The fifth section discusses the obtained results. The sixth and final chapter discusses future work.

Chapter 2 Need for Multiple Solutions

Many considerations go into the implementation of hardware designs. One of the last steps in the process is to realize the design either through fixed silicon in an application specific integrated circuit (ASIC) or programmed into a field programmable gated array (FPGA).

2.1 Power Considerations

The designs placed in FPGAs can be optimized for the specific application. An instance where the device is required to be portable and rely on a limited power supply, the limiting factor of the design is power. An example of this is a portable device. Such a device would rely on limited battery power to fuel it. In cases such as these, it makes sense to spend an extra amount of time generating a power-efficient solution in order to expand the in-use life of the product and reduce the recharges necessary for continued operation.

2.2 Delay Considerations

In cases where the processes performed by the circuit need to be performed multiple times rapidly, delay may be the limiting factor. An example lies in hardware cryptography, where the cipher is transmitted. If the cryptographer is fast enough, each partial syllable of each transmitted word can be encrypted in a different manner. While minimizing delay is always a good and necessary thing, under some circumstances spending extra time generating a circuit with less delay can be greatly beneficial.

2.3 Area Considerations

For area, manufacturing costs are the limiting factor. An example of this would be a small company developing a chip that is to be mass produced for market. By decreasing the physical area of the chip, the fabricator can place more chips on each silicon wafer. Since the cost of each wafer is fixed, the total cost of the individual chip is reduced and the yield per wafer is improved.

Chapter 3 Experimentation

In order to evaluate automation of design optimization, two computer aided design tool paths were selected. The first is Synplicity's Synplify Pro with Amplify. Synplicity's suite is for synthesis, placement and routing of hardware designs onto FPGAs. The second tool is Liberator. Liberator was developed as an automated design space exploration tool by Dr. Fuat Karakaya as part of his Ph.D. research at the University of Tennessee [1].

3.1 Synplify

Synplicity's Synplify Pro with Amplify is a powerful FPGA synthesis, placement and routing suite. The Synplify tool provides a full range of synthesis tools. Synplify allows the designer the ability to cross probe the synthesized code between critical paths and the register transfer level or the technology level. This control allows the designer the ability to identify critical paths and suggest synthesis constraints that allow the synthesizer to meet the design requirements. The design flow for Synplify is represented in figure 3.1.

The Amplify tool is an added package to Synplify Pro which extends the design constraint ability into the physical level, allowing the designer to specify which logic cells to use for certain areas of the design. The design flow for Amplify is represented in figure 3.2. Both of these design tools consider some of the designer constraints to be soft when the design requirements are not met. In other words, Synplify and Amplify use the designer constraints as a guideline in meeting the design requirements, and will give

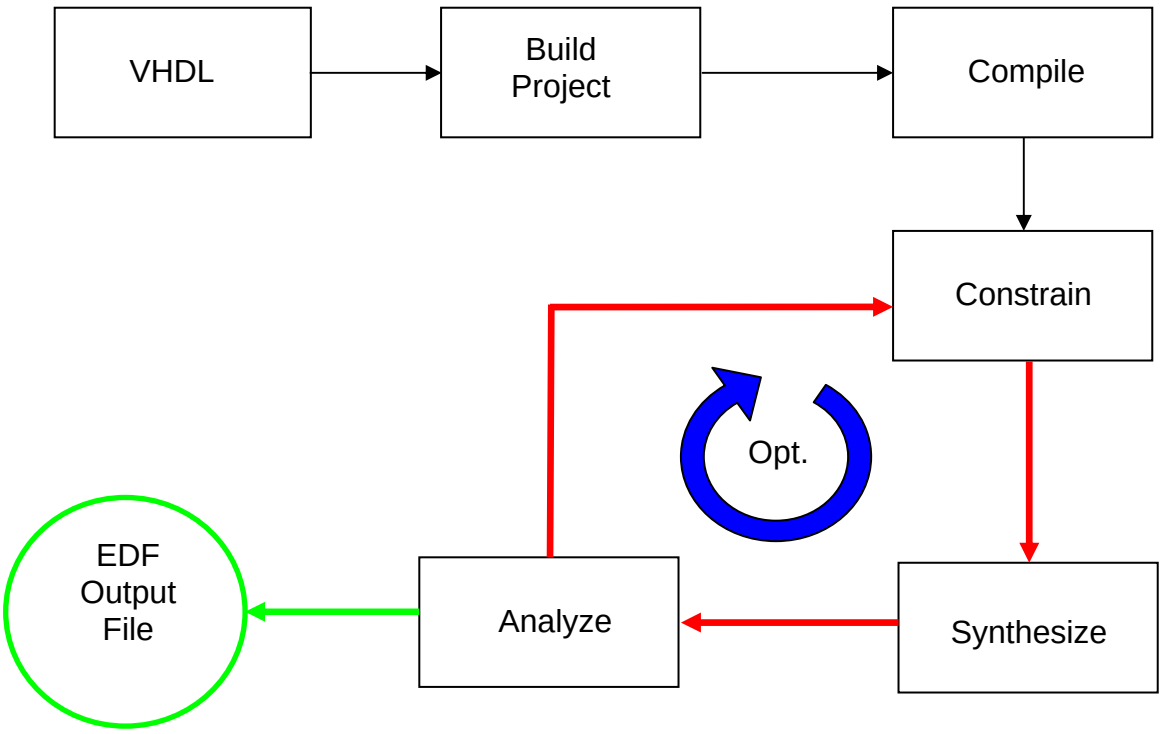

Figure 3.1: Synplify Design Flow

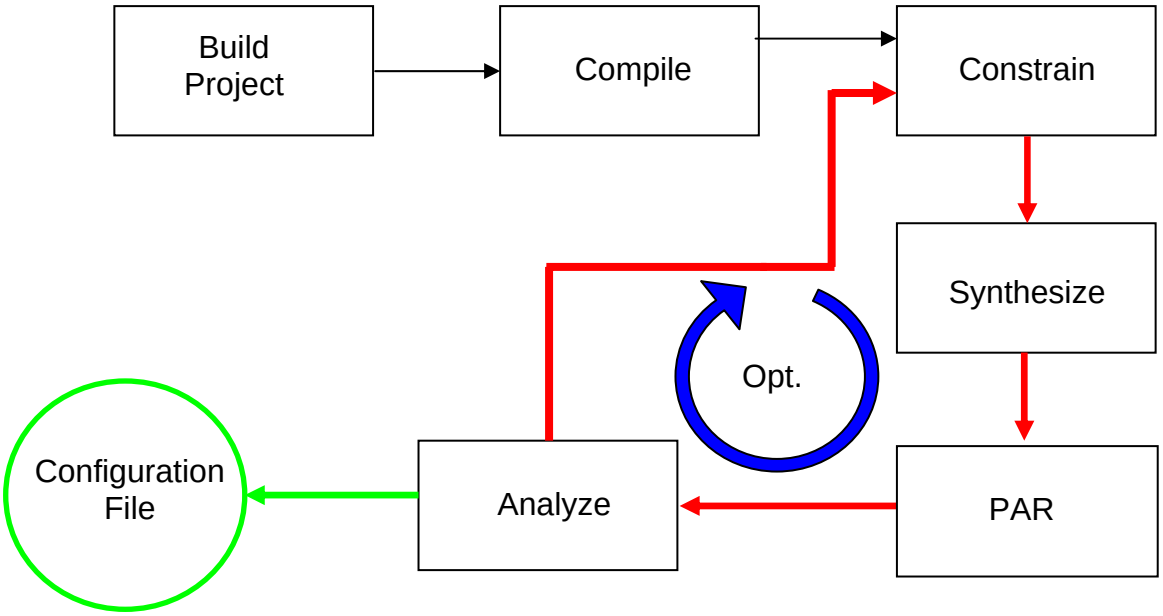

Figure 3.2: Amplify Design Flow

results outside of the constraints if the designer has made constraints that will not meet the design requirements.

Synplicity provides two methods for interaction with Synplify and Amplify. The first is the graphical user interface (GUI). The GUI for Synplify is most useful for the designer taking advantage of the timing and RTL or timing and technology cross probing. The typical view for exploring RTL can be found in figure 3.3. An example of cross probing by filtering the technology view down to a clock constrained critical path can be found in figure 3.4. For Amplify, the GUI can also be used to graphically assign physical constraints. An example of setting physical constraints can be found in figure 3.5.

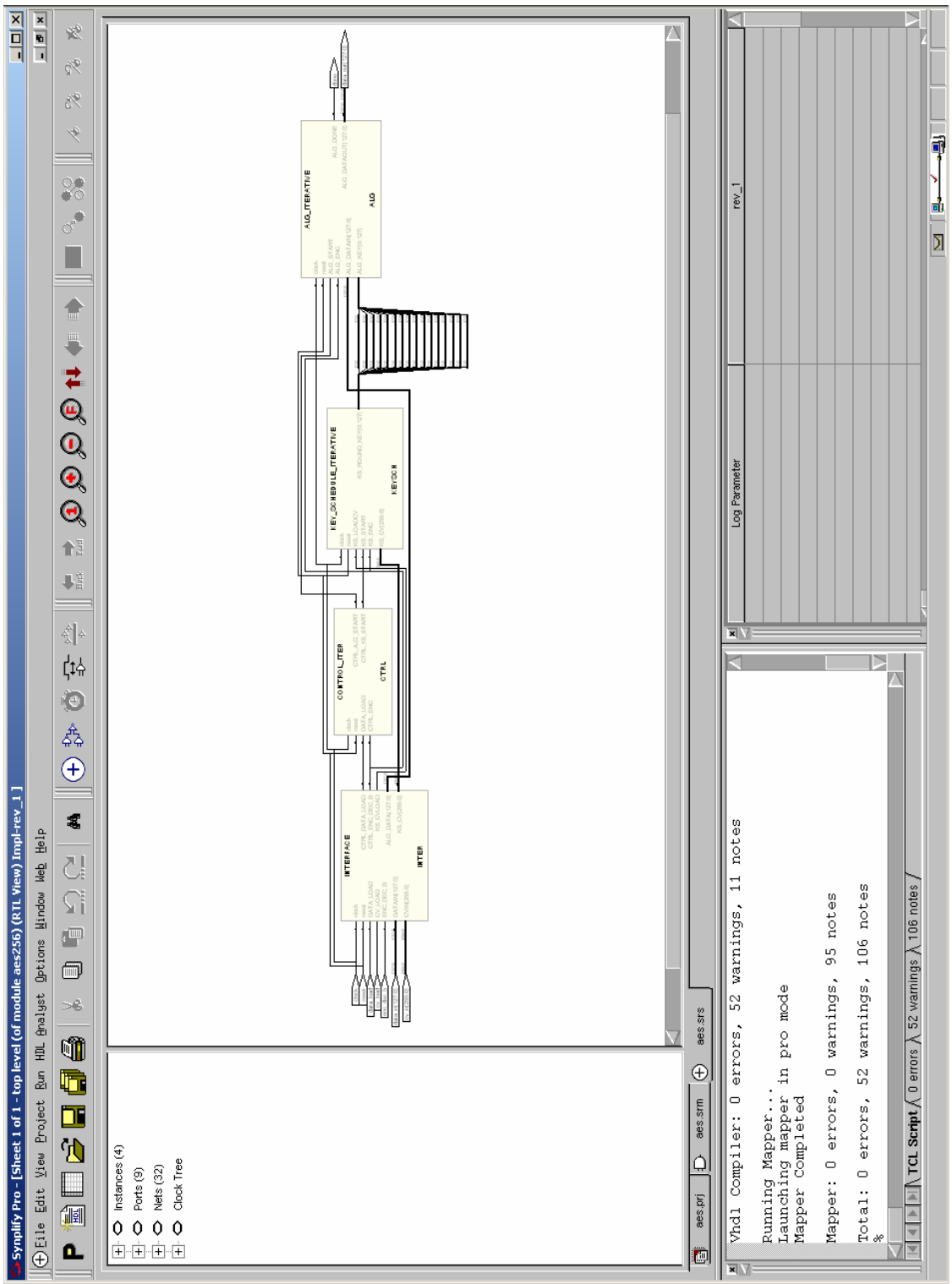

Figure 3.3: RTL View in Synplify

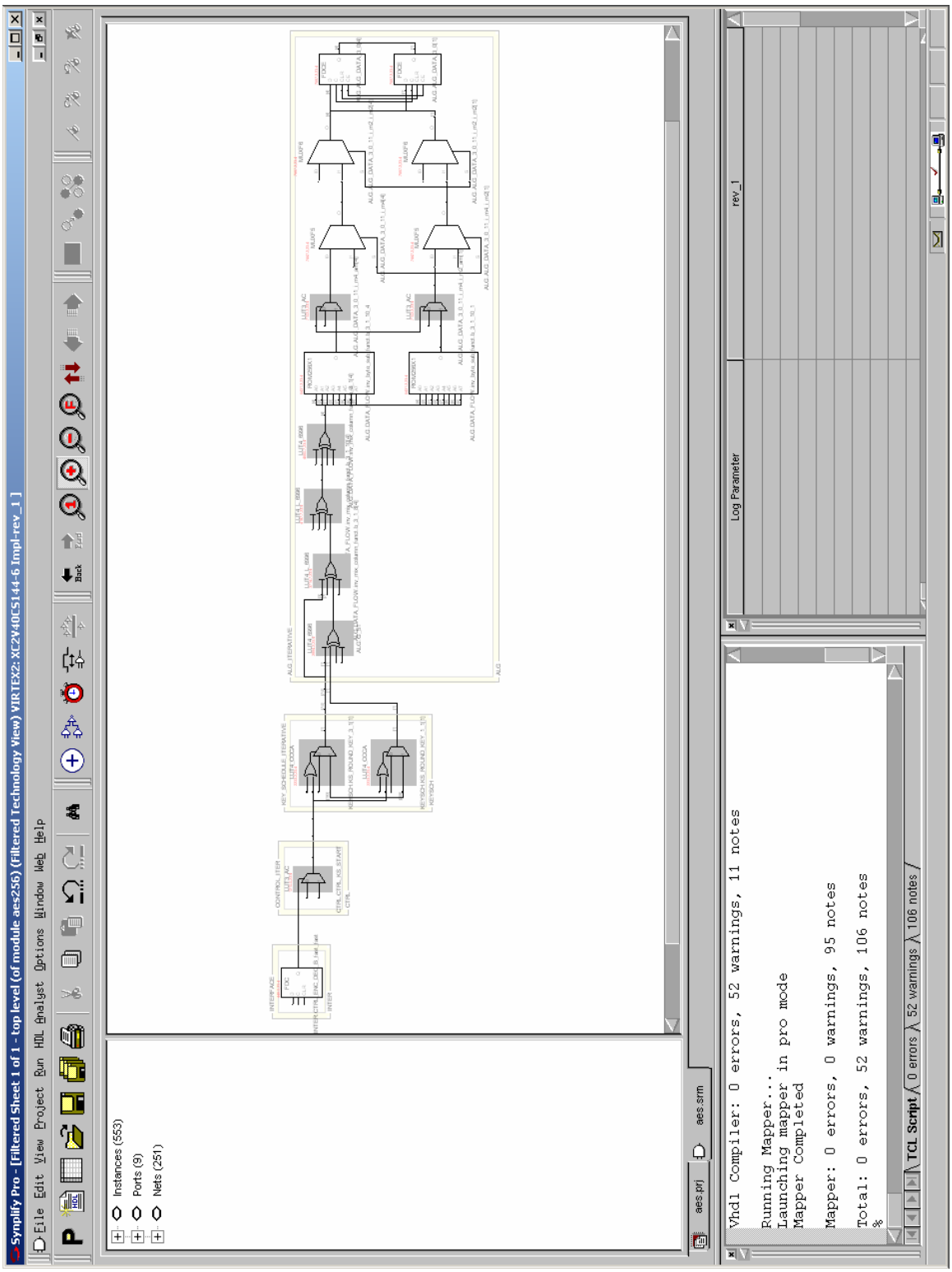

Figure 3.4: Filtered Critical Path Technology View in Synplify

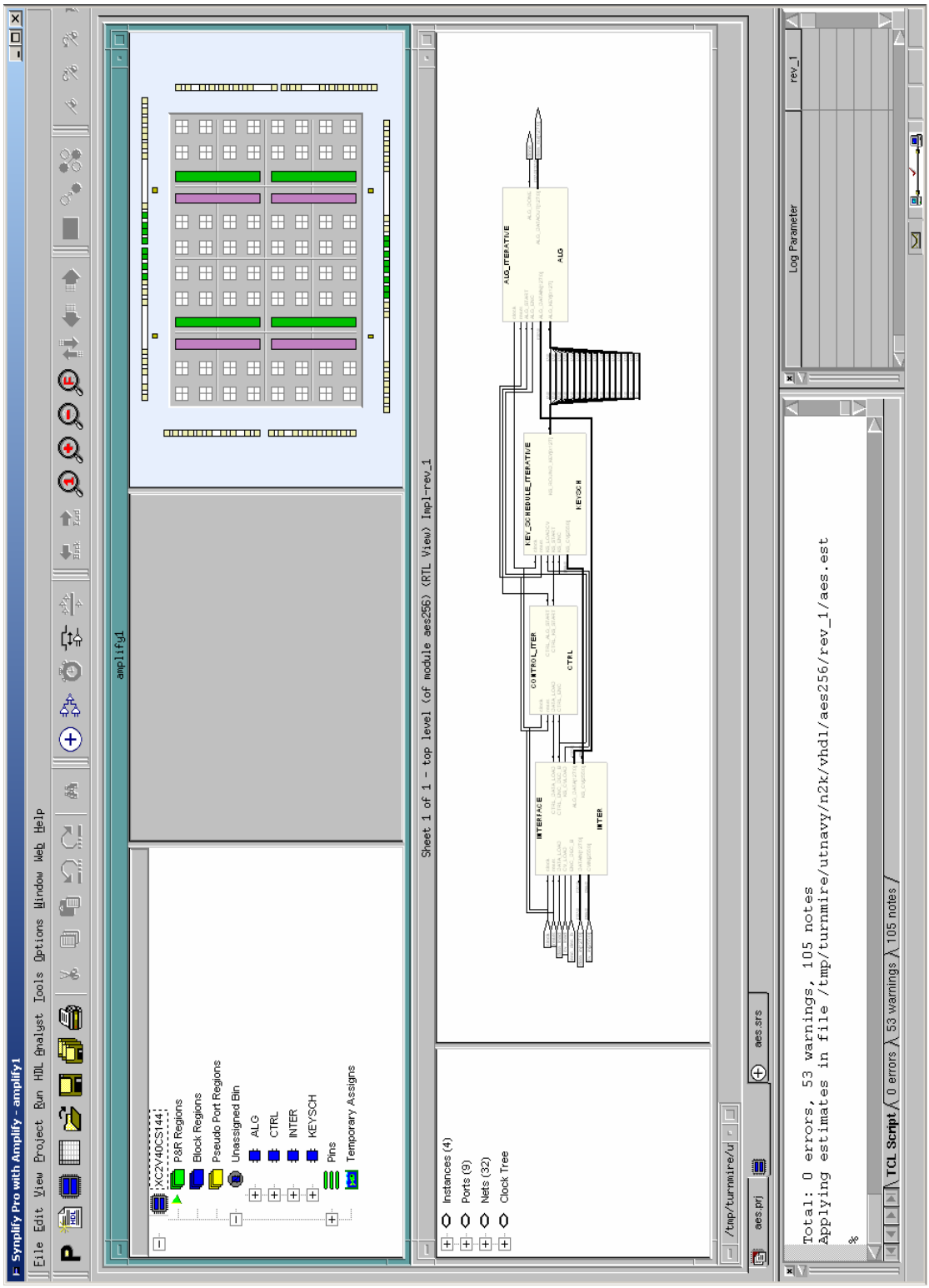

Figure 3.5: Amplify Physical Constraints GUI

The other method of interacting with Synplicity's tools is through a command line. All of the settings that are accessible through the GUI are available as settings in the command line environment. The easiest way to leverage the command line entry is through batch files or scripts. This allows the designer to make incremental improvements in the design without the need to be seated before a terminal or behind a high speed connection to the server running Synplify or Amplify.

3.1.1 Tutorials

The following tutorials were created for use in a computer-aided design of VLSI systems course. In 2005 and 2006, these tutorials were used by more than 30 students at The University of Tennessee to gain familiarization with Synplify Pro and Amplify. The files required were supplied by Synplicity, and can be found in the tutorials directory under the main install of Synplicity's tools. Other operations are detailed in the User's Manuals [4], [5] and [6].

3.1.1.1 A Synplify GUI Tutorial

Copy the tutorial files to a working directory and run Synplify Pro. Create a project by selecting *File -> Build Project*. Verify file type is set to HDL files, see figure 3.6, and click *Add All* then *OK*. Now that there is a basic project built, the file hierarchy needs to be set. Click the + next to the VHDL folder. Click *Run-> Arrange VHDL Files*. One can also manually do this by dragging and dropping the files into the correct order. The lowest level is at top of list to be processed first, highest at bottom to be processed last. Save the project by selecting *File -> Save* and name it tutorial.

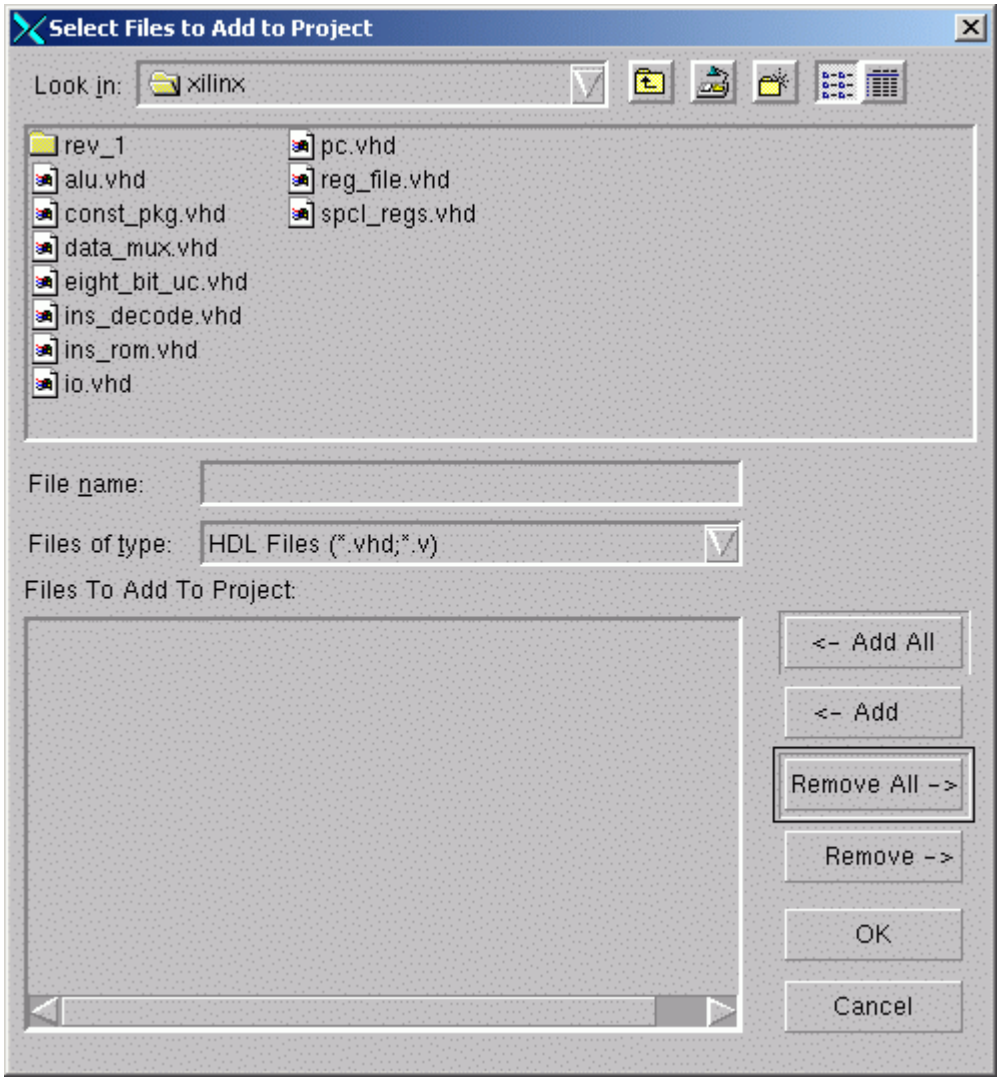

Figure 3.6: Adding Files to a Project

Check that the source compiles by pressing F7 or selecting *Run->Compile Only*. Review and correct the errors shown in the errors tab of the TCL window. Double clicking on the error line brings up the source to the line with the suspected error. Examine the RTL by clicking the *RTL* icon, shown in figure 3.7, to bring up a graphical high-level graphical model of the design. Use *zoom* to zoom in on a block and use *push/pop* to push into a block to see its internal workings. See figure 3.8. Close the RTL viewer.

To set constraints, start the SCOPE tool by clicking the *Constraints* icon or selecting *File->New->Constraint file*. Accept the default settings by clicking *OK*. Set the clock constraint by clicking on the clocks tab and enable the clock constraint by checking the enabled box next to clock. Enter a value of 170 in the frequency column and press Return, the remaining columns are filled automatically. Save the constraint file as “tutorial.sdc” and add the file to the project. Close the constraint editor.

Set the device parameters by clicking on the *Implementation Options* button, or choose *Project->Implementation Options*. Set the device to a Xilinx Virtex2 XC2V40-4 CS144. On the options tab select "Symbolic FSM Compiler" and "Resource Sharing", all others blank. On the Constraints tab make sure the constraint file (tutorial.sdc) is checked. On the Implementation Results tab, make sure the "Write Vendor Constraint File" option is checked. In the Timing Report tab, set the "Number of Critical Paths" to 25. Do not change anything under the VHDL tab.

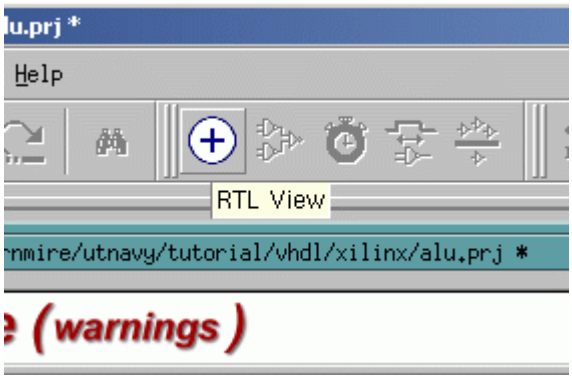

Figure 3.7: RTL Icon

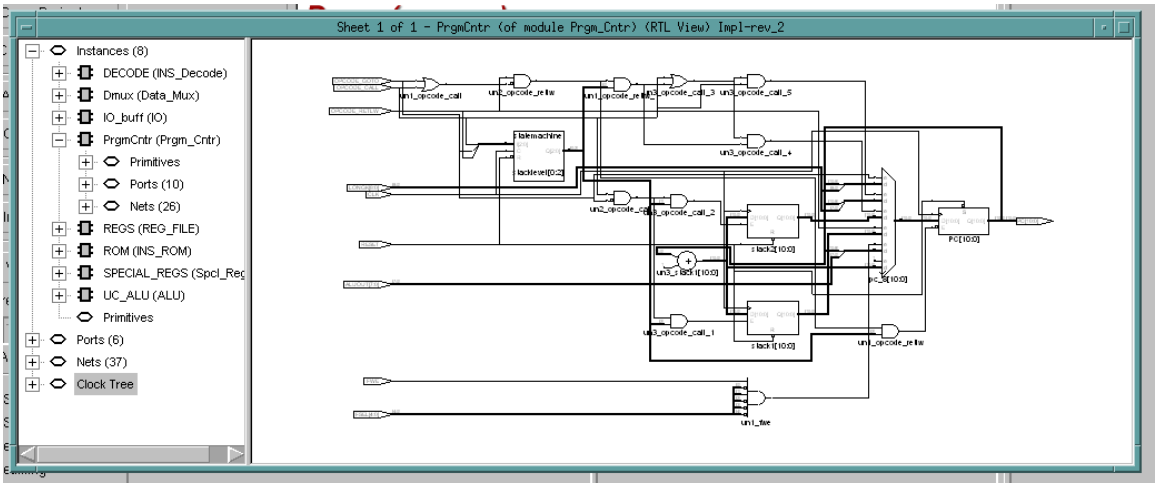

Figure 3.8: Design Exploration in RTL

Run synthesis by clicking the *Run* button in the project browser. To begin analyzing the results, start the Technology Viewer as shown in figure 3.9. This shows the design with cells directly mapped to the target technology. Browse the hierarchy on the left to see how selected primitives/modules are highlighted in the tech view.

To check the timing, open the Log Watch window by selecting *View->[check]Log Watch Window*. In the Log Watch window, select the first log parameter and select "Worst Slack" to display the worst slack as showing in figure 3.10. In the next two columns, use the pull down menu to display the estimated frequency and the requested frequency.

To see detailed information about critical paths, open the log file (eight_bit_uc.srr) by selecting the *view log* button. Scroll down to Performance Summary and Worst Paths Information. Use ctrl-f to open the search dialog and search for "worst path". This details the worst path, which will be made into a constrained critical path.

Analyze critical paths in the Technology View by opening the technology view window, figure 3.9, and select the critical path icon as shown in figure 3.11. Zooming in on the nodes of the critical path, you will notice that the slack (second number) is negative, meaning the timing criteria was not met. This will be resolved by setting cycle constraints and repeating the steps to synthesize the design. Repeating the synthesis process is known as resynthesizing. Figure 3.12 shows timing requirements met.

Figure 3.9: Technology Viewer

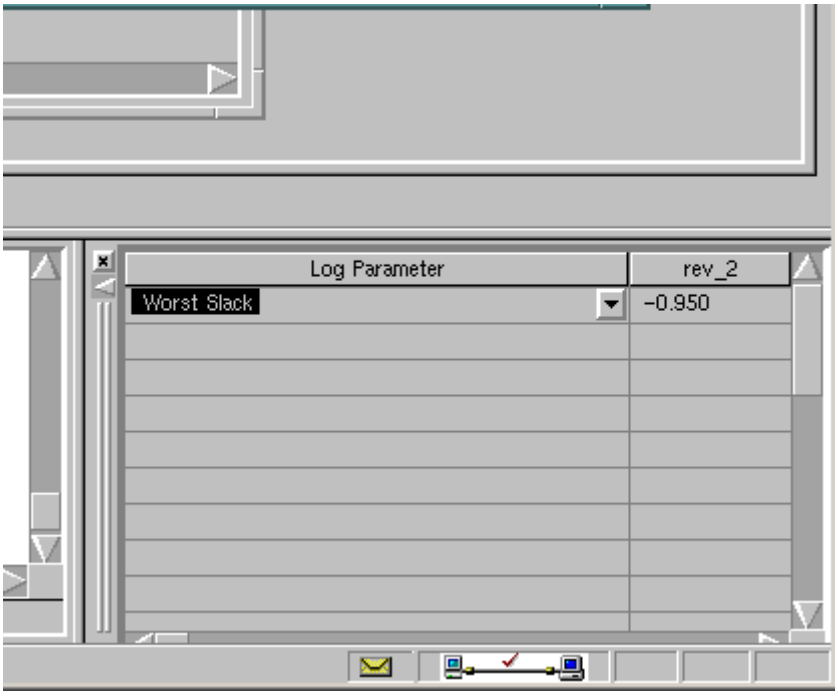

Figure 3.10: The Log Watch Window

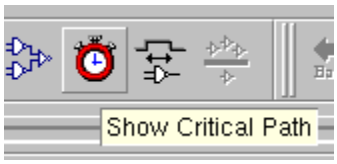

Figure 3.11: Critical Path Icon

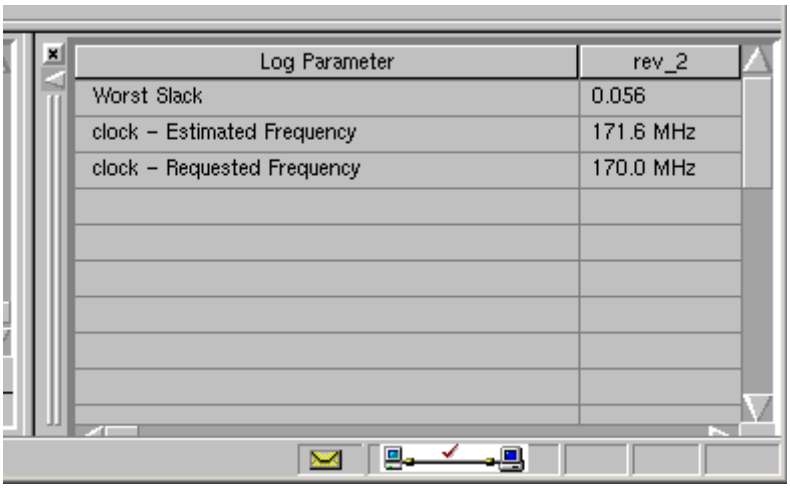

Figure 3.12: Timing Requirements Met

3.1.1.2 An Amplify GUI Tutorial

Copy the tutorial files into a working directory and run Amplify. Setup the project by selecting *Open Project -> Existing Project* and select "data_control.prj". Highlight "npc" and click *Impl Options*. Set the parameters for Device, Options, Constraints, Implementation Results, Timing Report, Verilog, Amplify and Netlist Restructure as shown in figures 3.13, 3.14, 3.15, 3.16, 3.17, 3.18, 3.19 and 3.20 respectively.

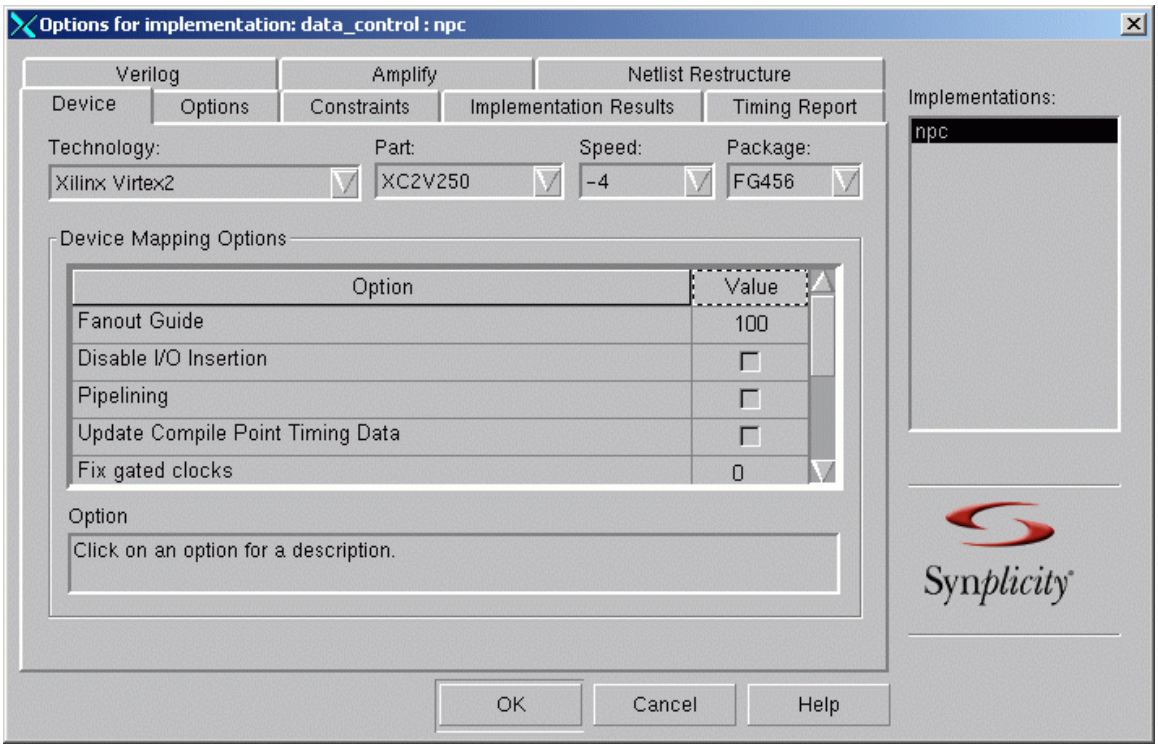

Figure 3.13: Device Parameters

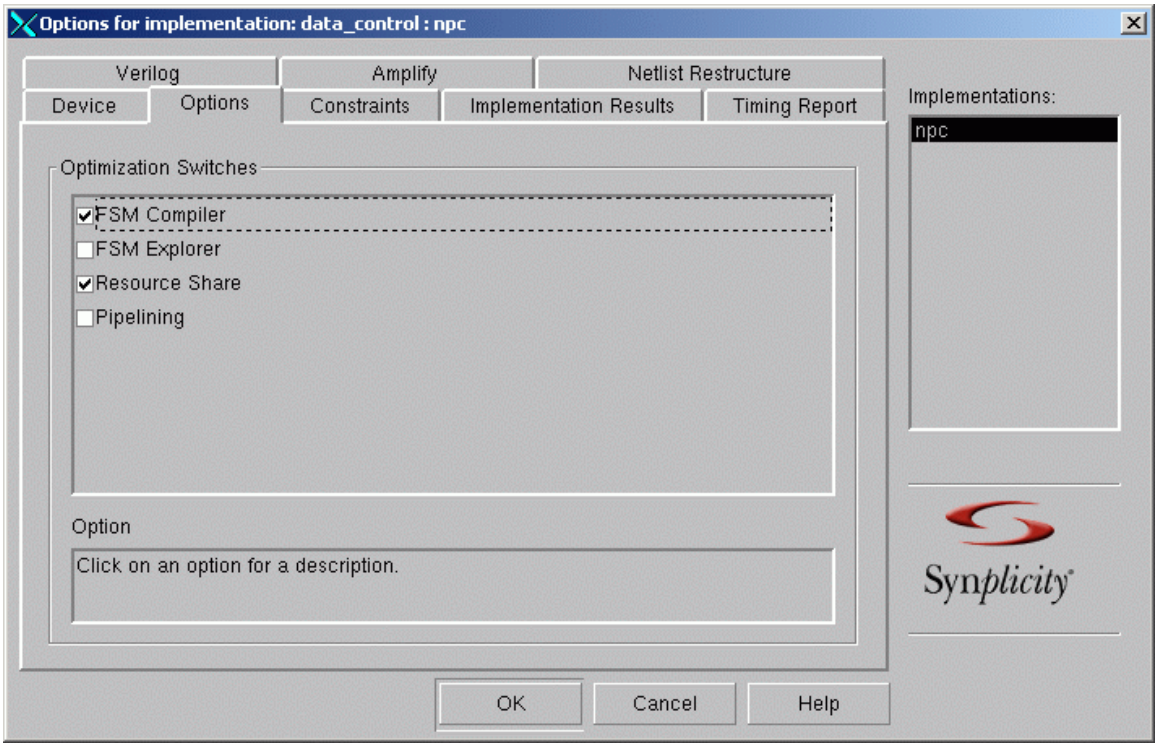

Figure 3.14: Options Parameters

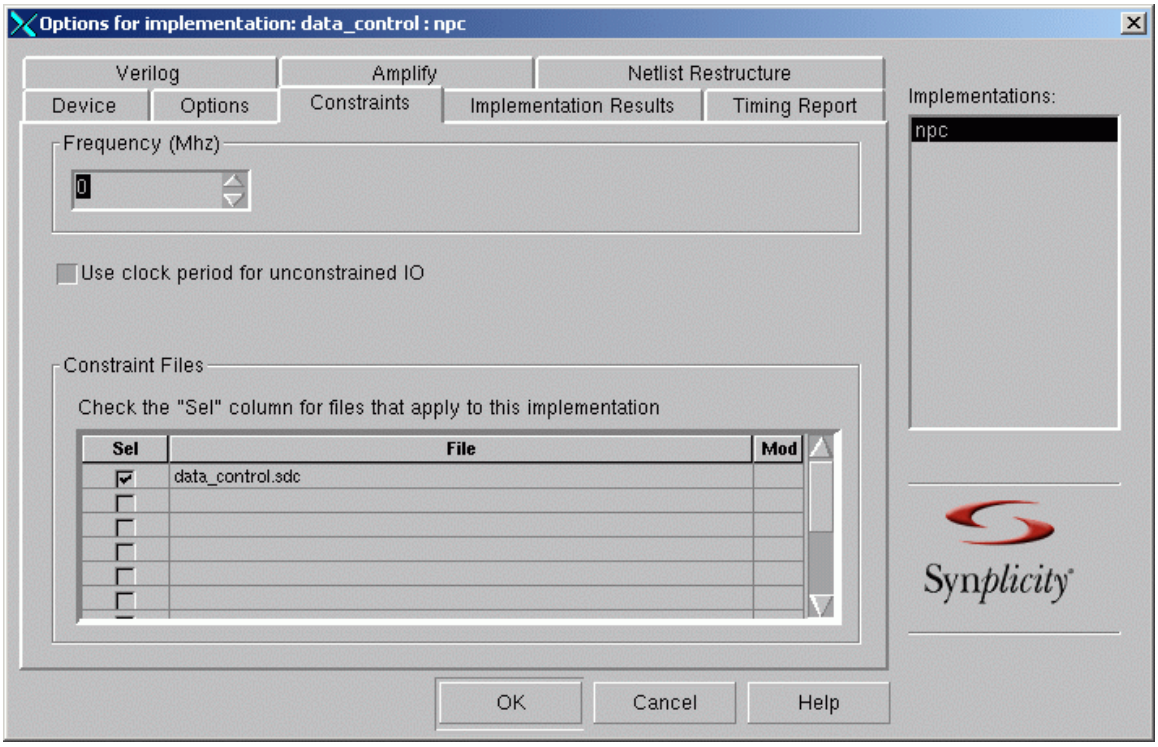

Figure 3.15: Constraints Parameters

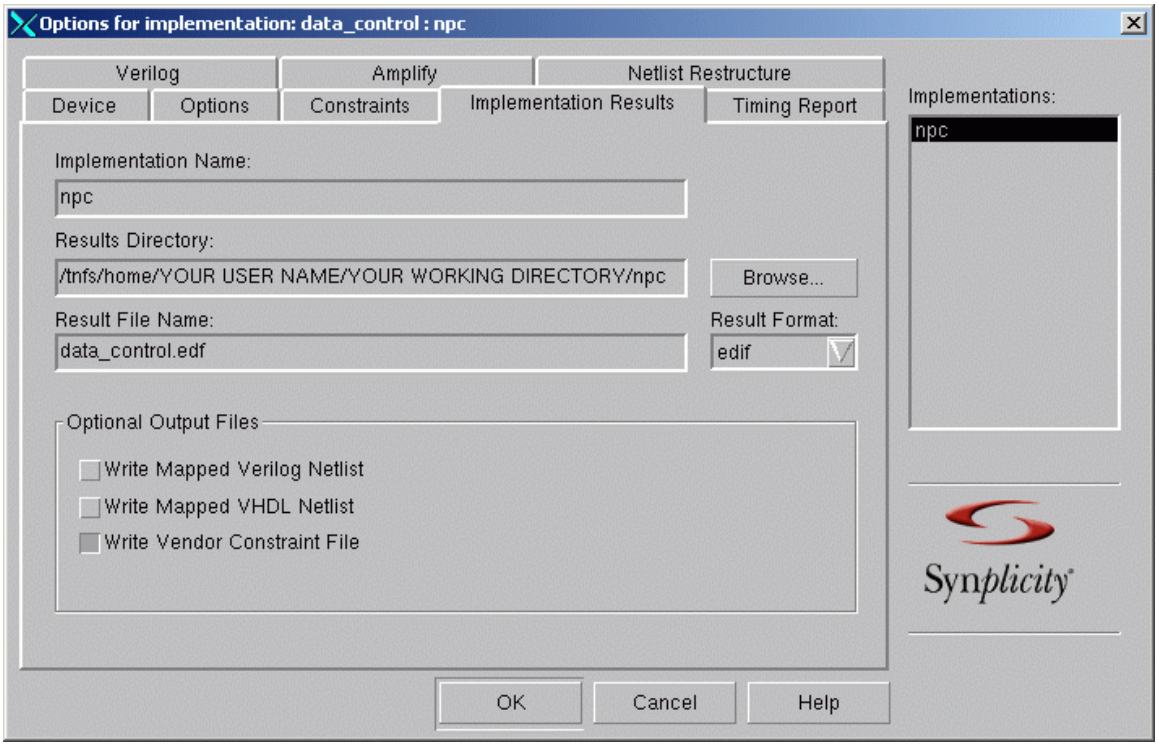

Figure 3.16: Implementation Results Parameters

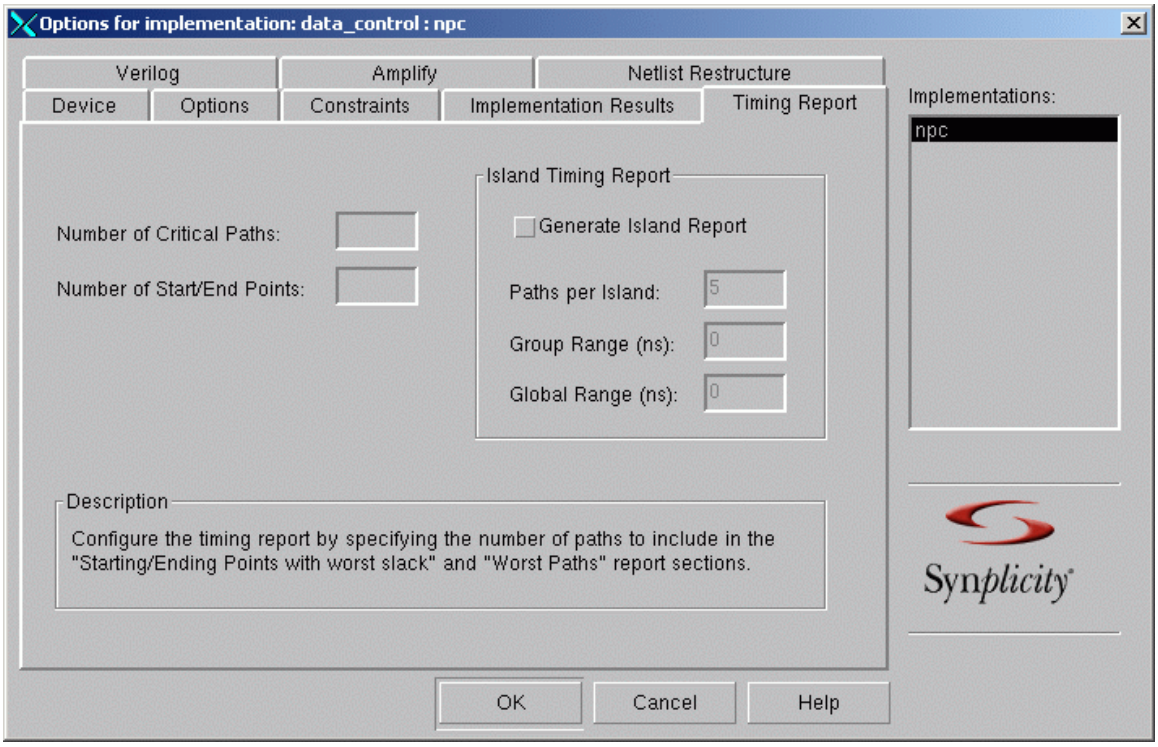

Figure 3.17: Timing Report Parameters

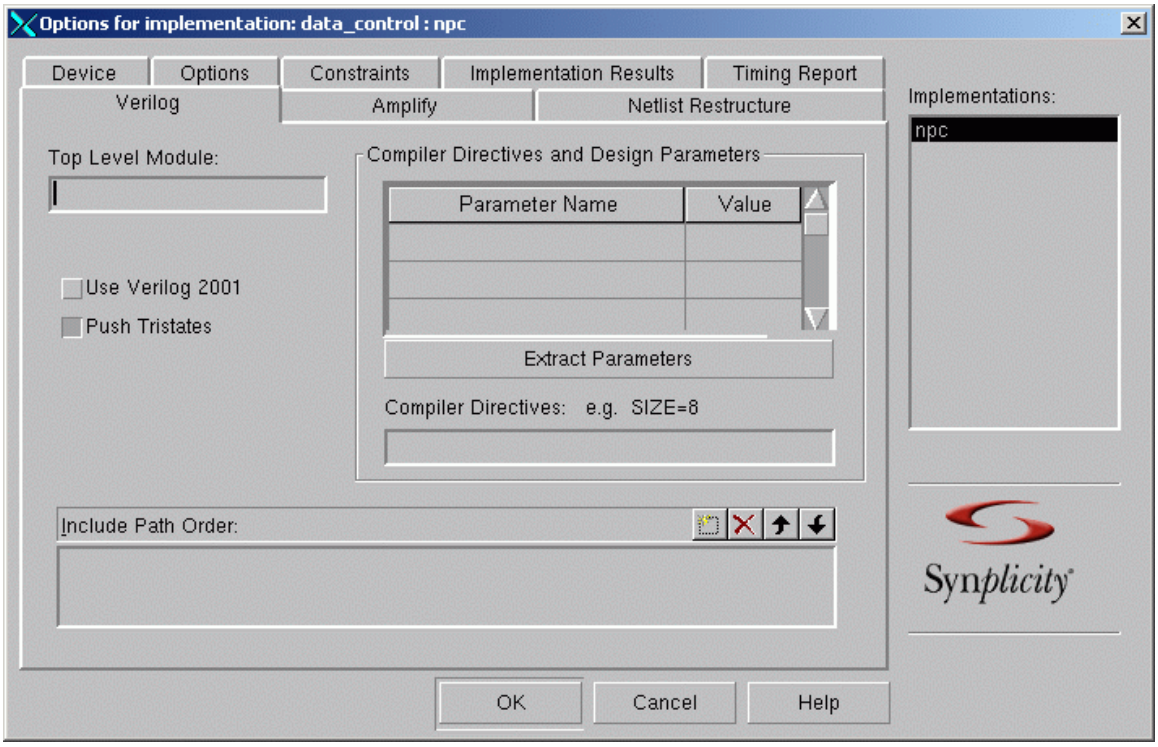

Figure 3.18: Verilog Parameters

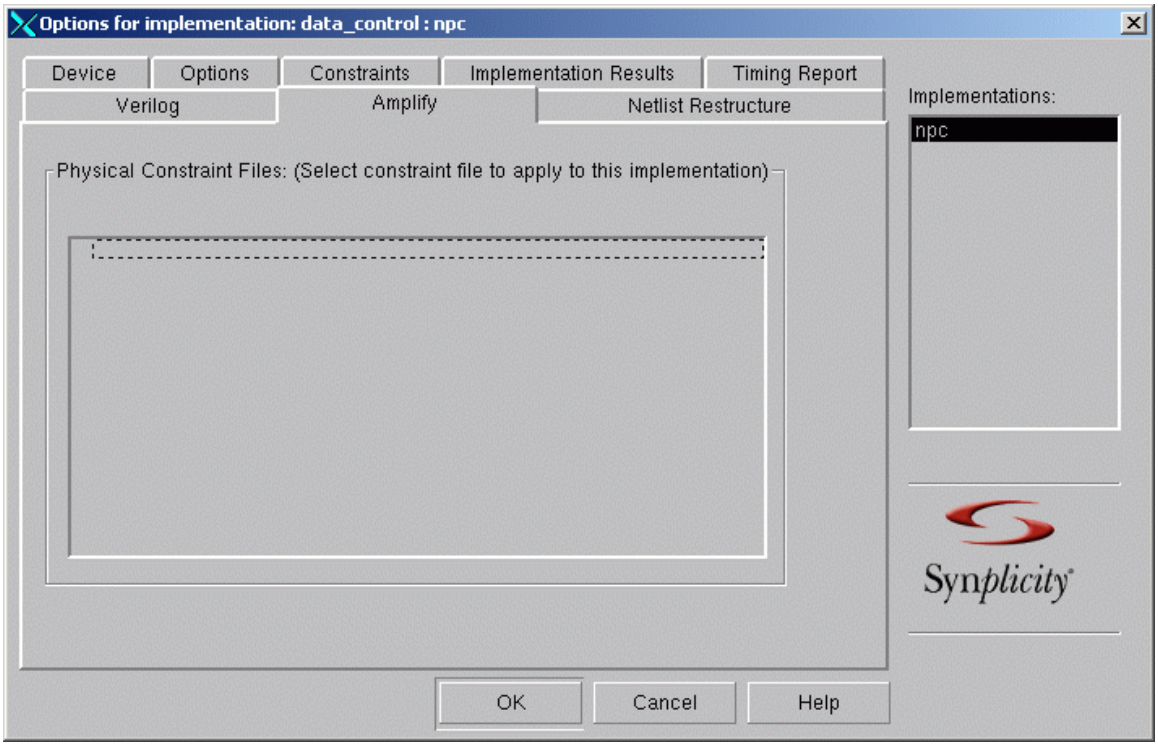

Figure 3.19: Amplify Parameters

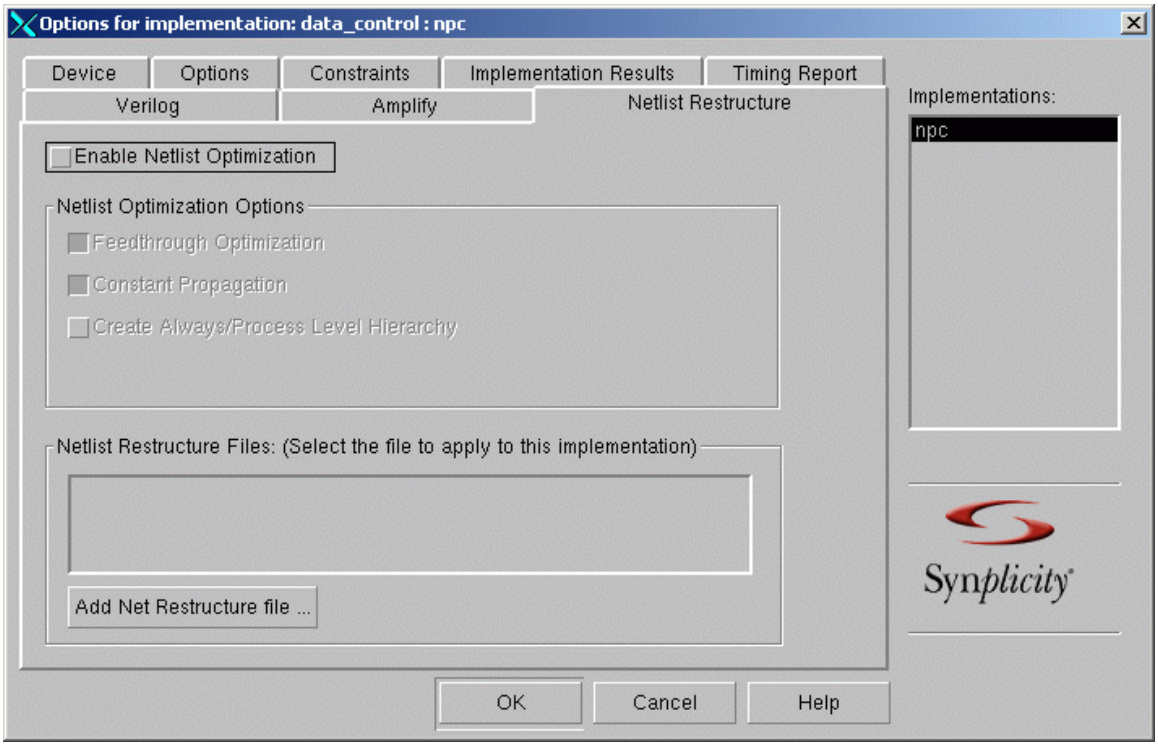

Figure 3.20: Netlist Restructure Parameters

Run synthesis without setting any physical constraints. Select *Run->Synthesize* from the menu, press f8, or press the large *Run* button.

Run the ISE Project Navigator placement and routing tool by highlighting the current implementation (npc) and click on *Options->Xilinx->Start ISE Project Navigator*. The tool automatically configures a project file. If your project has been updated since you last ran ISE Project Navigator you will need to update the file to continue by clicking *Yes* on the dialog box. With Amplify 3.7.1 using ISE Project Navigator versions 7.x and higher, it is necessary to open the project file that was created by selecting *File->Open Project,* highlighting data_control.ise and *Open*. Right click in the Sources in Project window and select *Add Source*. Add the data_control.edf file generated in the npc synthesis as shown in figure 3.21. Highlight *Implement Design* under *Processes for Current Sources*, right-click and select *Properties*. Select the *Map Properties* tab and verify that *Trim Unconnected Signals* is enabled. In the *Place & Route Properties* tab, set the *Place & Route effort Level (Overall)* to "*Normal*". Verify that *User Timing Constraints* is selected. In the *Post-Place & Route Static Timing Report Properties* set the *Timing Report (Number of Items)* to "*100*". Click *OK* to save the settings. Highlight *Implement Design in the Processes for Current Source* window, right-click and select *Run*.

Verify the Placement and Routing by expanding Implement Design then expand Place & Route. Double click on *Place & Route Report* to view the report. Right-click *Generate Post-Place & Route Static Timing* and select *Run* to generate the timing report. View the timing report and find the start and end points for the worst path. If there is no

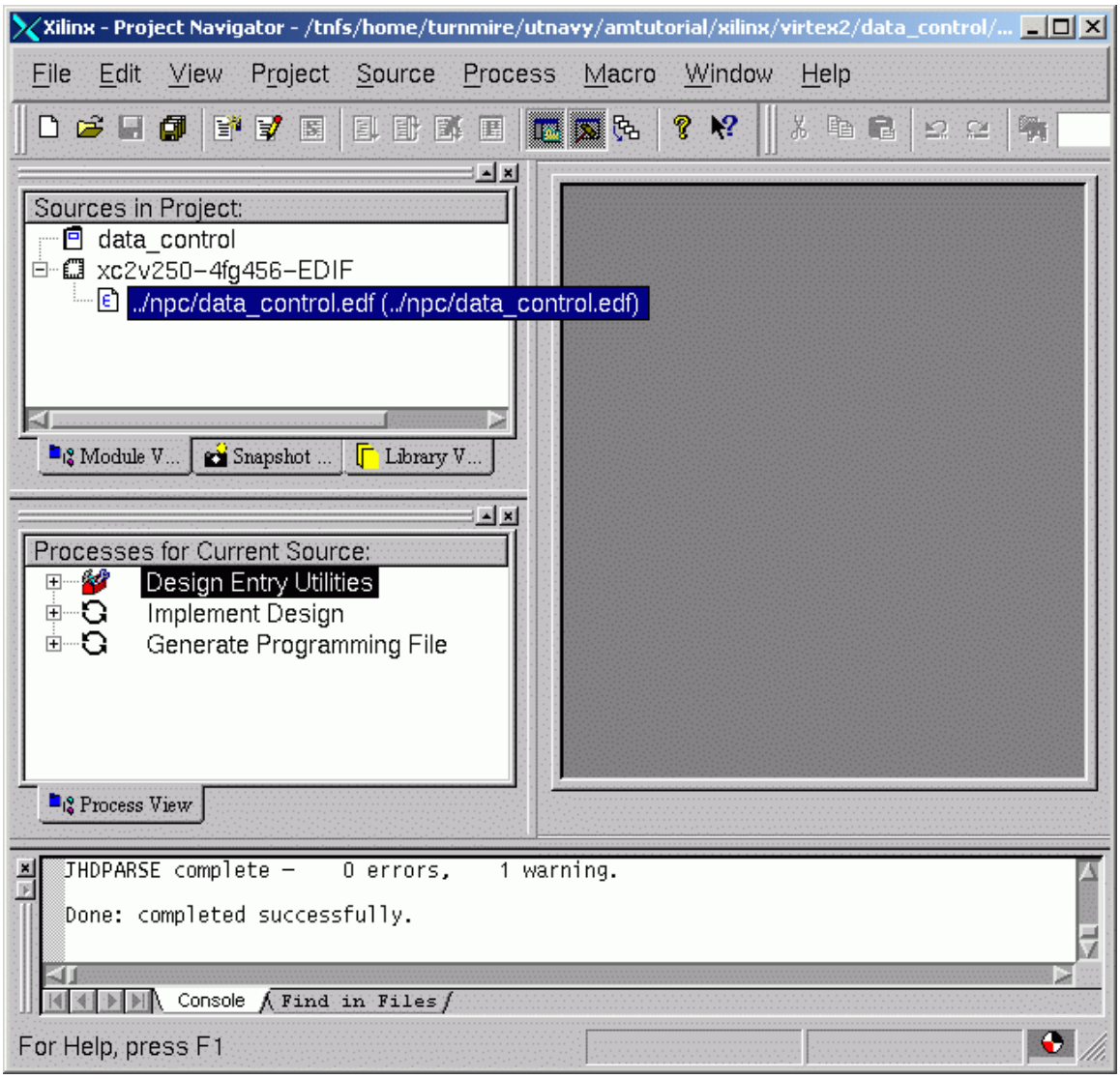

Figure 3.21: Adding data_control.edf to ISE Project Navigator

worst path indicated, go back to the Amplify window and edit the data_control.sdc constraint file setting the clock constraint to a higher value.

Start a new implementation with physical constraints by clicking on the *New Impl* button in the Amplify Project view. Change the implementation name under Implementation Results to "pc". Click *OK* to save the changes. Select *File->New --> Physical Constraint File*. Check "Add to Project". Enter the name "amplify" and check the file location. Click *OK* and select "No" for the estimation file dialog box.

Create a Region for Critical Paths by right clicking in the regions view and select *Add-> Block Region* and left click and drag from the upper left CLB at (20,29) to the lower-right CLB at (29,0). This creates a CLB region of 30x10 like that of figure 3.22. Right click in the RTL view and select *Find*. To find the path, type in the start and end points of the critical paths discovered in the Timing Analyzer. Click close when done. Notice the critical paths are highlighted in the RTL view. Select *HDL Analyst -> Filter Schematic*. Right click in the RTL view and select *Expand Paths*. Right Click and select *All Schematic -> Instances*. Right Click and select *Assign to -> rgn1*.

Run Estimate Regions by clicking on Block Regions in the Physical Tree view. Right click in the Physical Report View and select *Show/Hide columns*. Verify that Area, Area Use, Area Use (%) and Name are selected. Right Click on the device and select *Estimate All Regions*. Click *File -> Save All*.

To run synthesis with physical constraints, from the project view, make sure pc is selected. Click on *Impl Options* to update the implementation options. Under Amplify, make sure the Physical Constraint Files box is checked and the path points to the

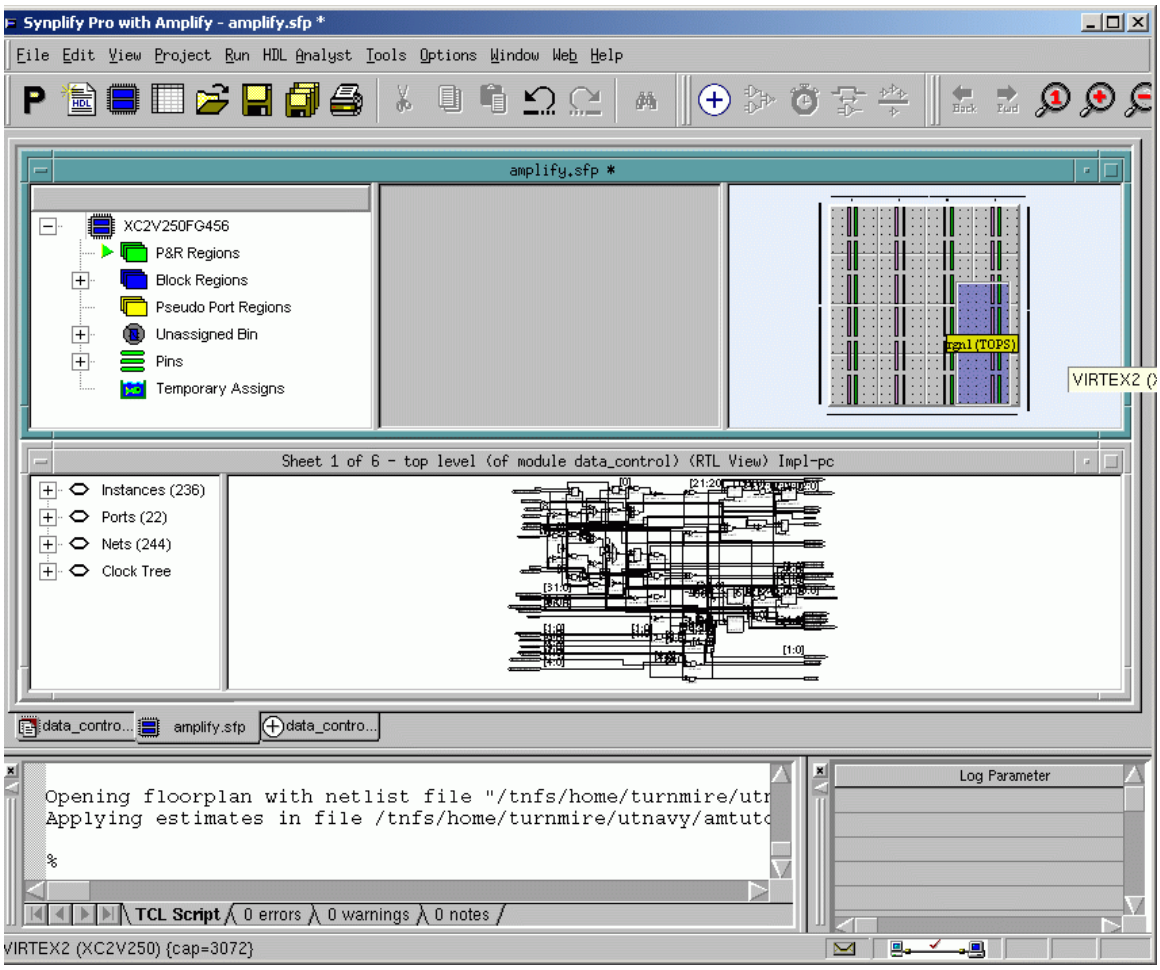

Figure 3.22: Creating a CLB Region

amplify.sfp file you created. Click *OK* to accept the options and select *Run* to synthesize the project.

Rerun the Xilinx Placement and Routing by running the ISE Project Navigator placement and routing tool and select the edf file in the pc directory. Reanalyze the timing results.

3.2 Liberator

Liberator is a GUI front end to seven computer aided design tools designed to automate the optimization process for application specific integrated circuits (ASICs). Liberator was developed by The University of Tennessee. Liberator makes use of the Design Space Search Algorithm (DSSA) developed by Dr. Fuat Karakaya [1]. Pseudo-code for the DSSA can be found in figure 3.23. The ASIC design flow including the DSSA optimization loop can be found in figure 3.24. The GUI of Liberator is shown in figure 3.25.

Liberator is designed to work on a select group of end-user-configurable circuits [3]. These templates can be found and configured under the Macrolist button in the Liberator GUI. The following circuits are available: Adder, Multiplier, Complex Multiplier, FFT and FIR. Once the circuit to be optimized is configured, the DSSA can be started by selecting the Synopsys Power Compiler button. The DSSA allows a number of options for configuration, including technology to target, run time stop condition, minimum improvement ratio stop condition, step size and field of measure. Figure 3.26 illustrates the configuration interface.

```
eps:=min. improvement;
s:=iteration_step_size;
 n:=number_of_iterations;
 r:=run_time;
 i:=improvement_ratio;
 f:=figure_of_merit;
Label0: Loop {
 if (noi==0) {
run(defaultflow.scr);}
 else {
 run(optimizationflow.scr); }
 P:=extract_P(power.rpt);
 D:=extract_D(timing.rpt);
 A:=extract_A(area.rpt);
Path:=extract_max_path(timing.rpt);
VP:=extract_violater_path(violater.rpt);
FOM(noi):=calculate_fom(f,P,D,A);
 if (noi == 0) {
save_design_parameters(0,P,D,A,FOM(0)); }
If (VP==0) {
impr:=calculate_impr(FOM(0),FOM(noi));
save_design_parameters(noi,P,D,A,FOM(noi),impr);}
const(noi)=generate_new_constrain(Path,D,P,A,s);
update_synthesis_script(const(noi));
 noi++;
 if (eps > (impr(noi)-impr(noi-1))) \{terminate;}
 if (impr => i) { case:=1; goto Label1; }
 if (cput => r) { case:=2; goto Label1; }
 if(noi => n ) { case:=3; goto Label1; } }
 else {
vconst(noi)=generate_constraint_for_violater(VP,const(noi));
 update_synthesis_script(vconst(noi)); }
 } while(noi < n);
 Label1: print "Continue?";
 answer:='yes'/'no';
 if (answer:='yes') {
 if (case==1) \{i:=new_i; }
 if (case==2) {r:= cput + extra_r; }
 if (case==3) {n:=noi+ extra_n; }
 goto Label0; }
 else { terminate; }
```

Figure 3.23: DSSA Algorithm

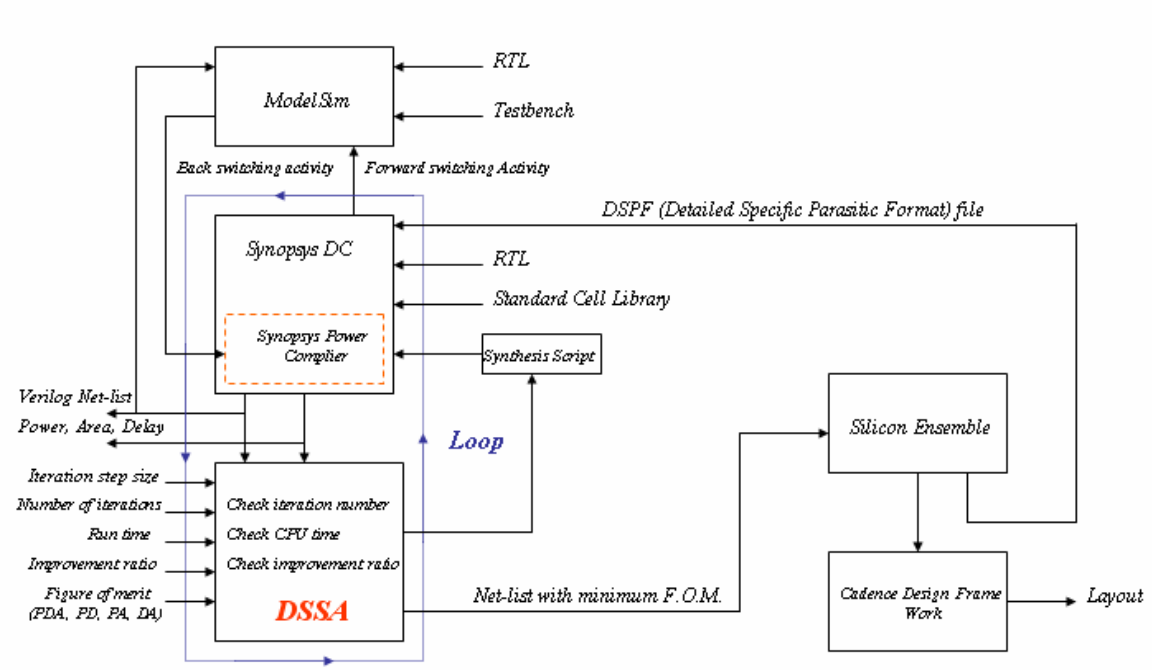

Figure 3.24: ASIC Design Flow with DSSA Loop

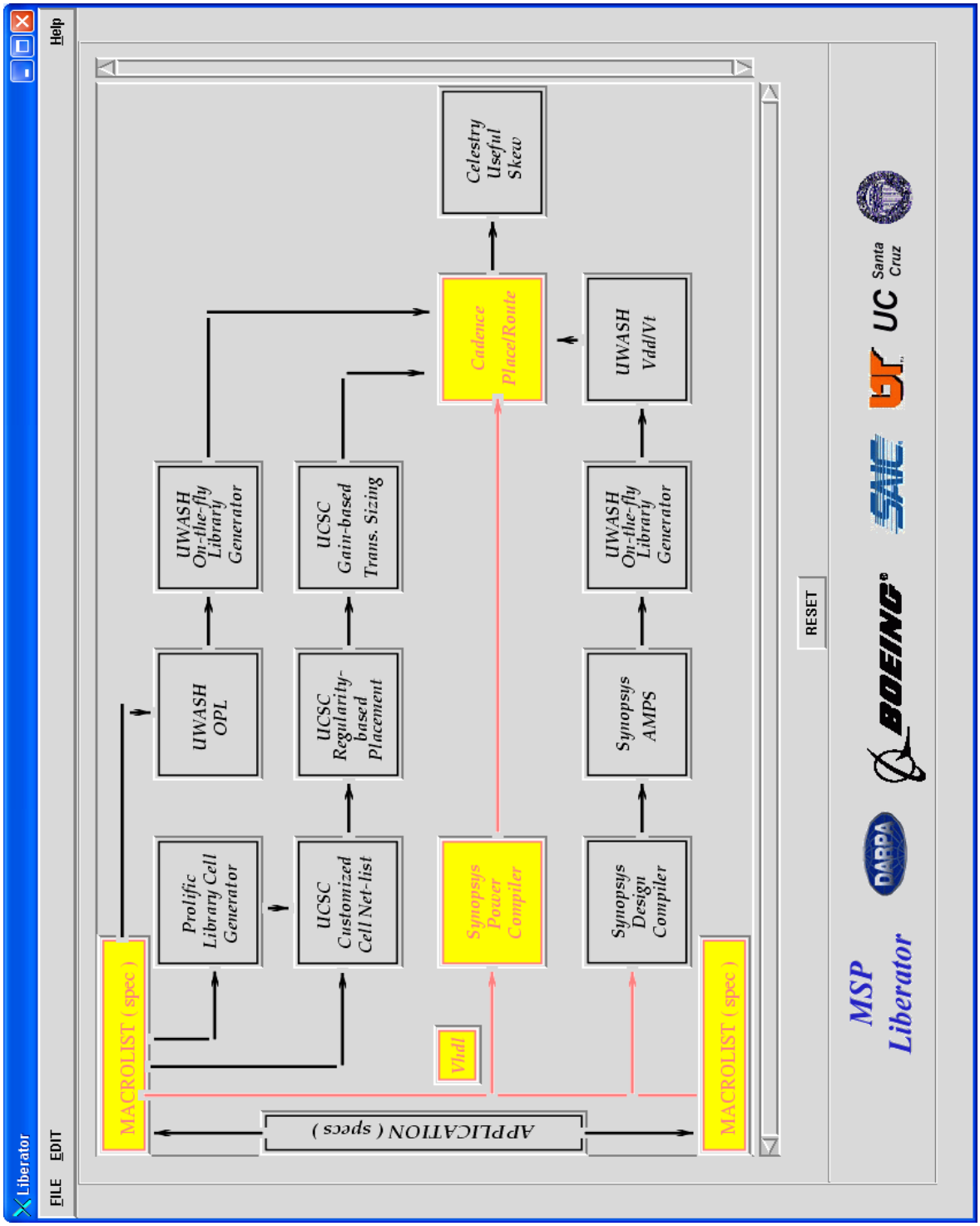

Figure 3.25: Liberator GUI

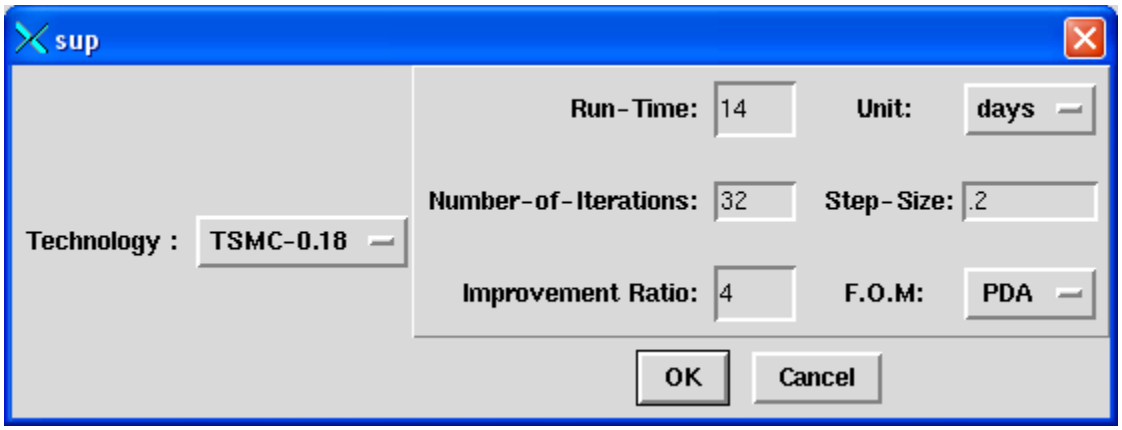

Figure 3.26: DSSA Configuration GUI

Chapter 4 Results

Evaluation of the software tools described above was performed by running optimization methods upon various designs. A single design that shows typical results was selected for Synplify, while Liberator was exercised against all the various circuits that are included.

4.1 Synplify

In order to properly evaluate the design space available to Synplify, a script was written to test a design over a range of possible configurations. An example script can be found in figure 4.1. Upon running this script, the manner in which Synplify adheres to constraints is clear. Even without enabling the options for retiming and pipelining, Synplify searches these areas of the design space to find a solution closest to the specified design requirements. Pipelining is the process of dividing a process into stages that can be overlapped, similar to an assembly line. Figure 4.2 is a plot of the estimated frequency of an AES256 core on a Xilinx Virtex-II Pro XC2VP30-6 versus requested frequency. Note that the requested frequency is scaled logarithmically. The AES256 code used for this example was developed by Scott Fields and Adam Miller [2].

The results shown in figure 4.2 are typical for Synplify, Synplify Pro and Amplify using similar scripts. There are two distinct regions in the returned solutions: when the first pass meets or exceeds the design requirements and when the design fails to meet the design requirements. In the latter case, Synplify uses retiming and pipelining, regardless of the setting, in an attempt to meet the design requirements.

```
# synplify_pro -batch scriptname
project -load /path/project.prj # Load Project
impl -name /path/rev_2 # Set implementation revision path
set_option -technology VIRTEX2P # Set technology
set_option -part XC2VP30 # Select Part
set_option -grade -6 # Select Speed Grade of Part
set_option -pipe 0 # Tell Synplify to not attempt pipelining
set_option -retiming 0 # Tell Synplify to not attempt retiming
set_option -num_critical_paths 50 # Set the number of critical paths in
timing report
set_option -num_startend_points 50 # Set the number of start and end
points in timing report
set freqs {
        1
        2
        10
        20
        100
        200
        1000
        2000
} # Set the range of frequencies to target
foreach frequency $freqs {
 set_option -frequency $frequency # Set the frequency for each
iteration
 set_option -result_file /path/rev_2/$frequency.no.edf # EDF file for
each frequency
 project -log_file $frequency.no.srr # Create separate report for each
frequency
 project –run
}

set_option -pipe 1 # Tell Synplify to attempt pipelining
set_option -retiming 1 # Tell Synplify to attempt retiming
set freqsy {
        1
        2
        10
        20
        100
        200
        1000
        2000
} # Set the range of frequencies to target
foreach frequencyy $freqsy {
 set_option -frequency $frequencyy # Set the frequency for each
iteration
 set_option -result_file /path/rev_2/$frequencyy.yes.edf
 project -log_file $frequencyy.yes.srr # Separate log & EDF files per
frequency
 project -run
}
```

Figure 4.1: Sample Script for Synplify

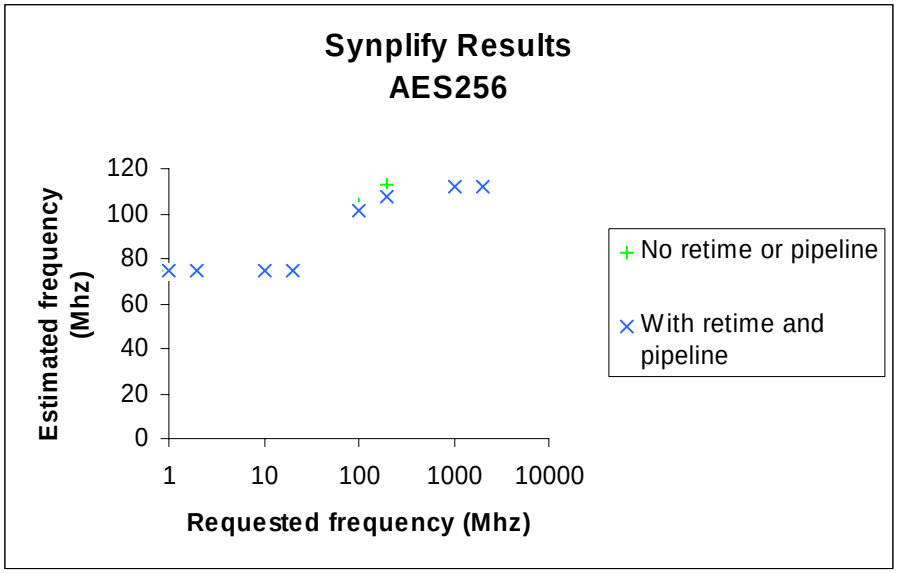

Figure 4.2: Example Synplify Results

The amount of chip area utilized varies approximately 1% between the two regions, from 13013 to 13384 look up tables. Power was estimated for the 1MHz and the 200MHz requested frequency solutions. A switching activity file was generated by tracing a cursory simulation in Model*SIM* and then fed into Xilinx XPower along with the placed and routed solution. The 1MHz design yielded an estimated dynamic power use of 188.68mW while the 200MHz design is estimated at 187.43mW. The power estimations may be inaccurate in depiction of the final operating power, but clearly express the relative difference in power usage between the two designs [7].

4.2 Liberator

The DSSA was invoked for each of the five circuits in Liberator. The same parameters were used to configure the DSSA in each case. The targeted technology was set to TSMC-18. Run time was set to 14 days. The number of results was set to 32. The improvement ratio was set to 4. The step size was set to 0.2. The field of measure was set to PDA. The 64-bit FFT stopped after 14 days of run time with 30 data points generated. The remaining modules stopped after generating 32 data points. The initial point is generated using no optimization settings. In every module, the second data point is the minimum power point. From the second point, the delay is reduced by the step size for each iterative point. The 32-bit FIR filter was the only module to reach the end of the design space and begin oscillating between the two points of lowest delay. The 64-bit FFT obtained a minimum area point at the 16^{th} iteration, where all the other modules had minimum area at the initial point with no optimization specified. The results for each module are plotted in various ways to best display the path DSSA followed in optimizing the design. The initial point as well as the minimum delay, minimum power and minimum area points are highlighted in the PDA vs. Delay plot for each module. The results obtained for a 32-bit adder are plotted in figures 4.3 through 4.6. The results obtained for a 32-bit multiplier are plotted in figures 4.7 through 4.10. The results obtained for a 32-bit complex multiplier are plotted in figures 4.11 through 4.14. The results obtained for a 64-bit FFT are plotted in figures 4.15 through 4.18. The results obtained for a 32-bit FIR are plotted in figures 4.19 through 4.22.

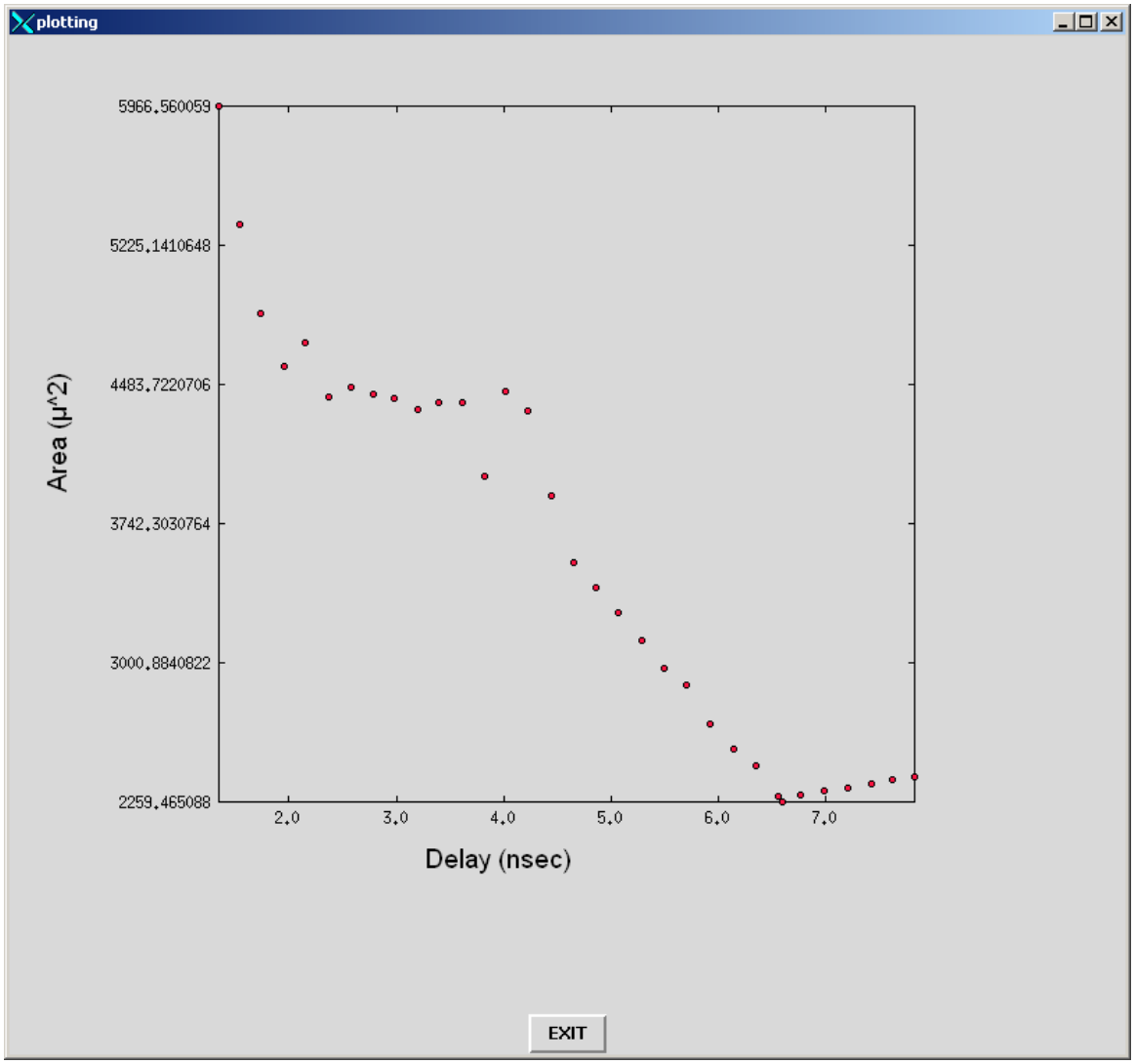

Figure 4.3: Area vs. Delay for 32-bit Adder

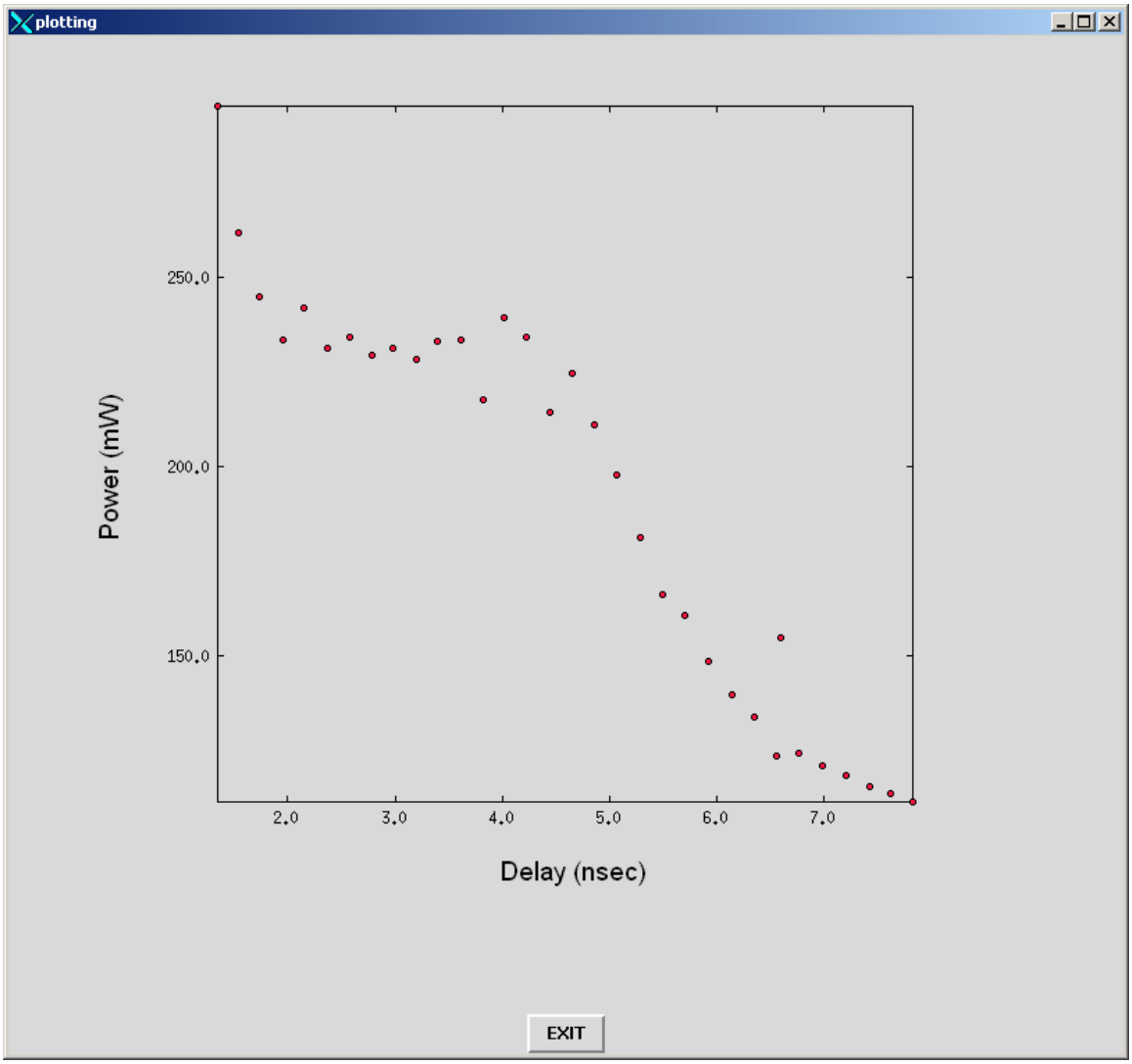

Figure 4.4: Power vs. Delay for 32-bit Adder

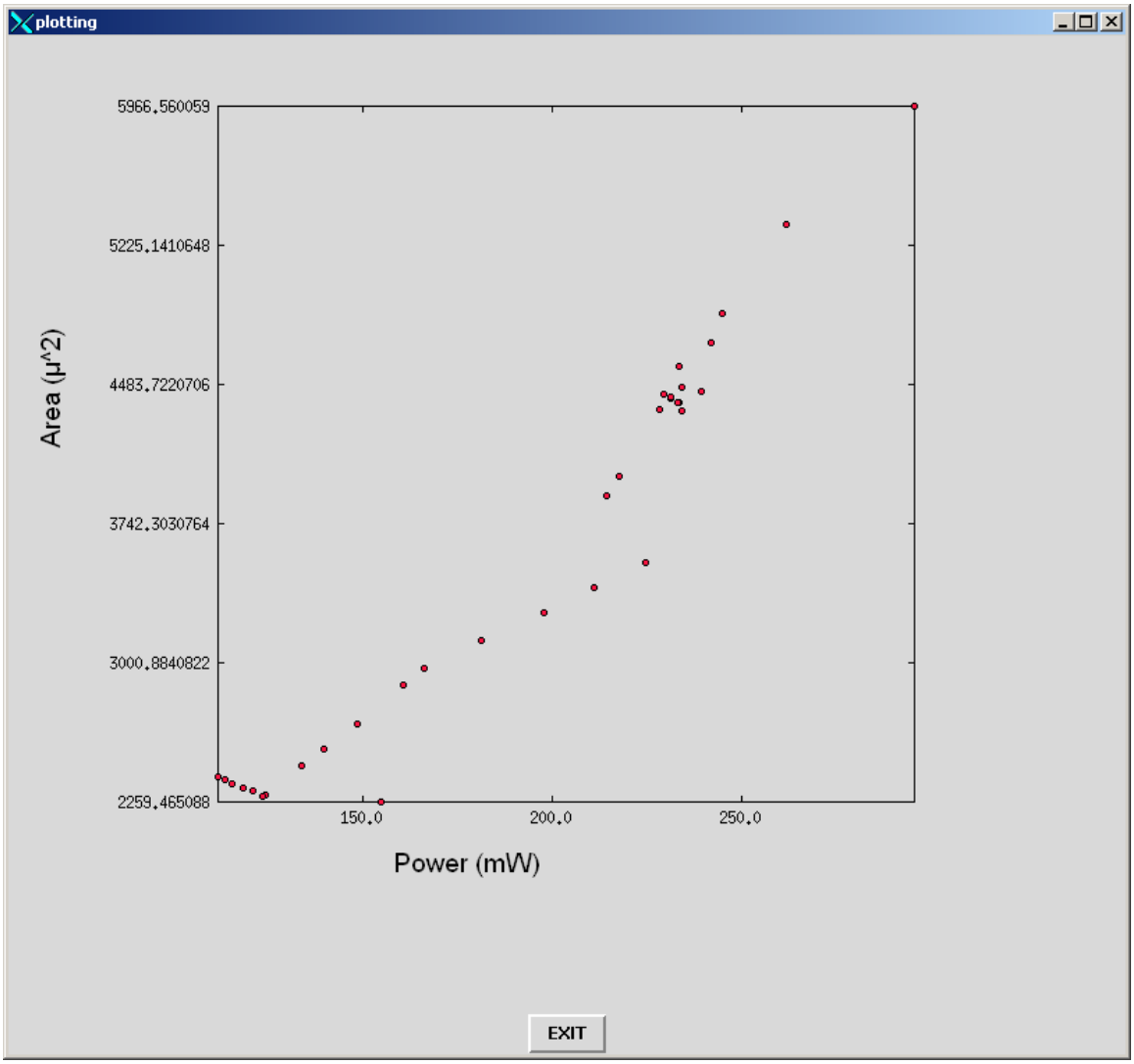

Figure 4.5: Area vs. Power for 32-bit Adder

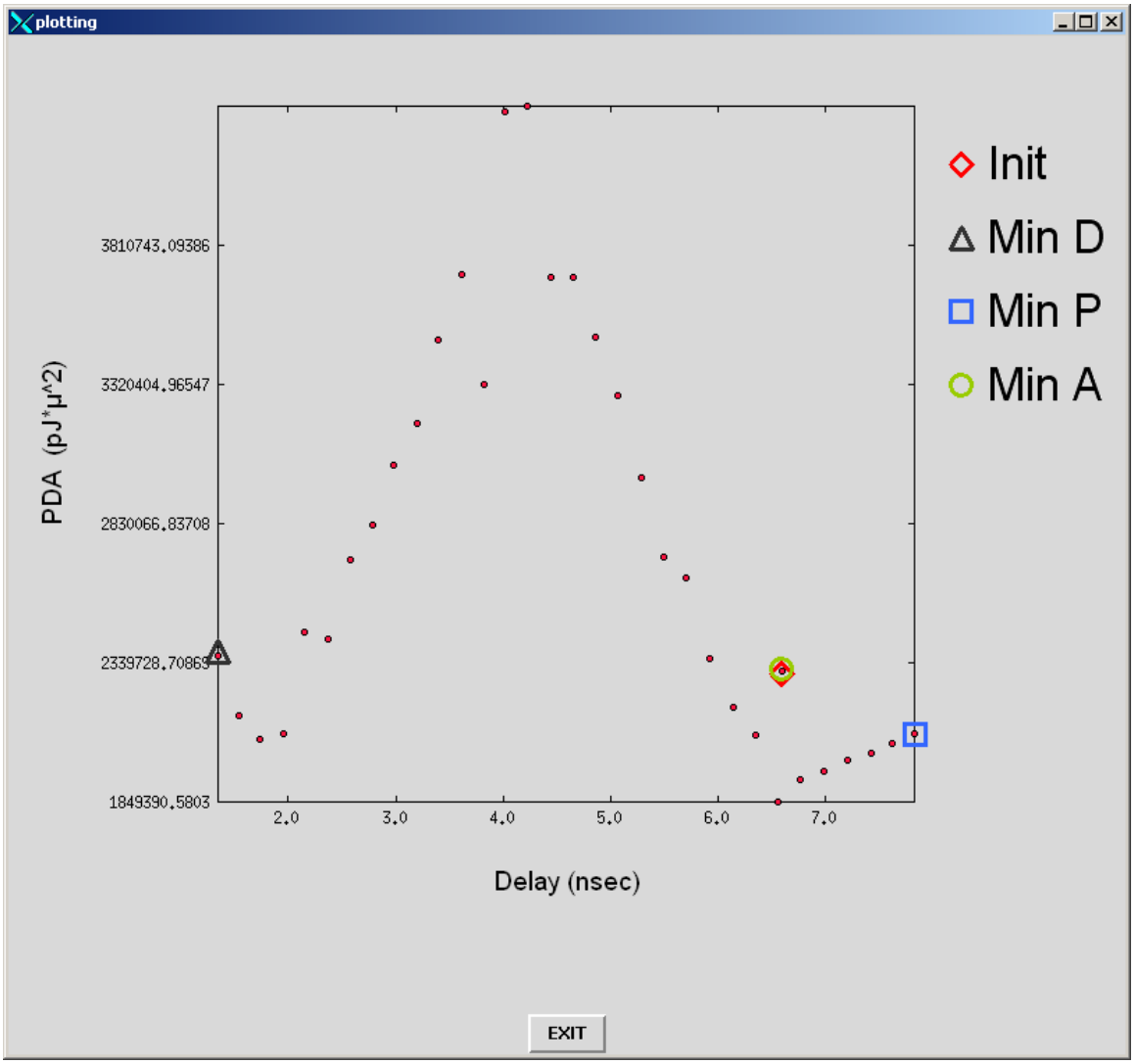

Figure 4.6: PDA vs. Delay for 32-bit Adder

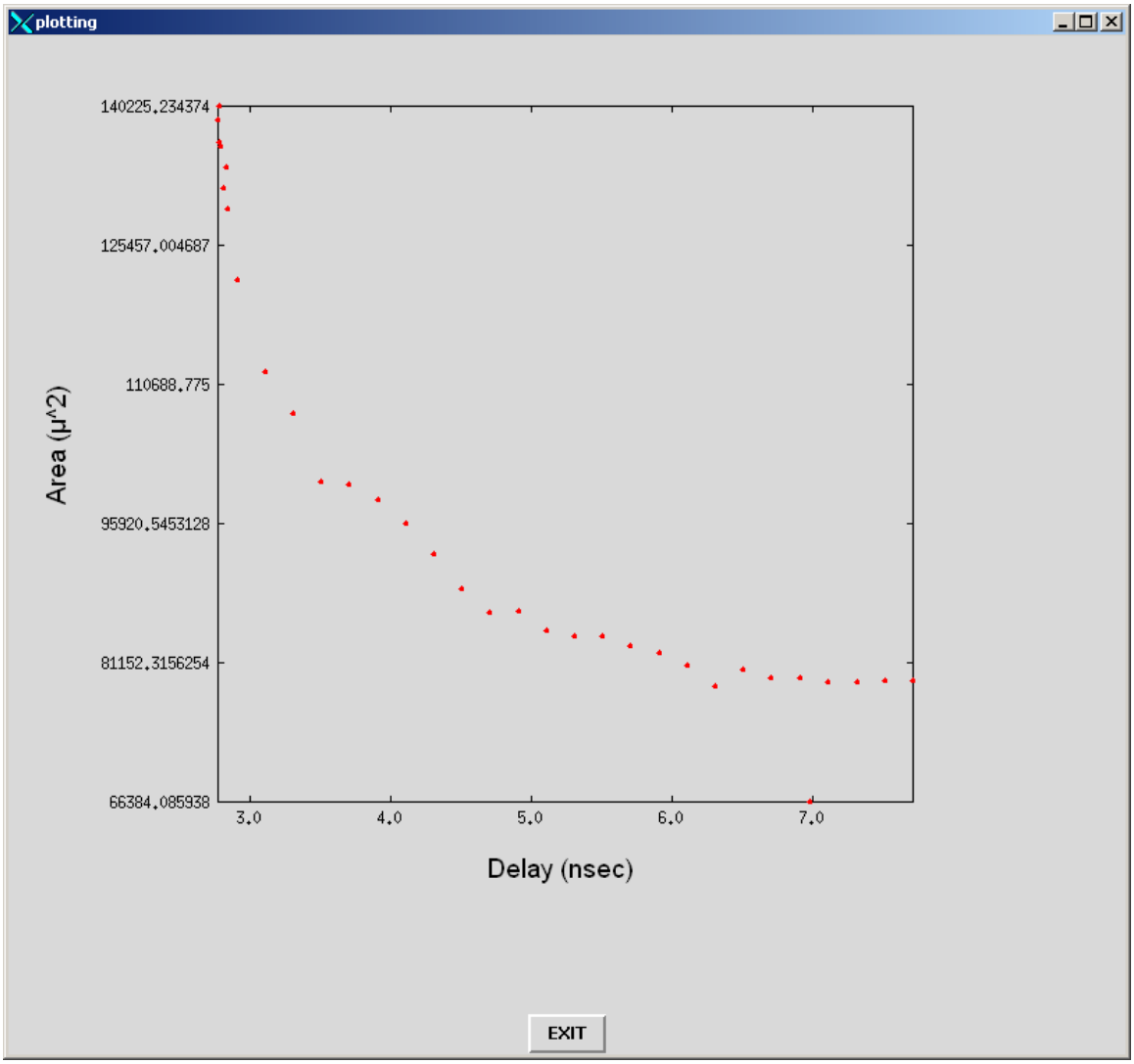

Figure 4.7: Area vs. Delay for 32-bit Multiplier

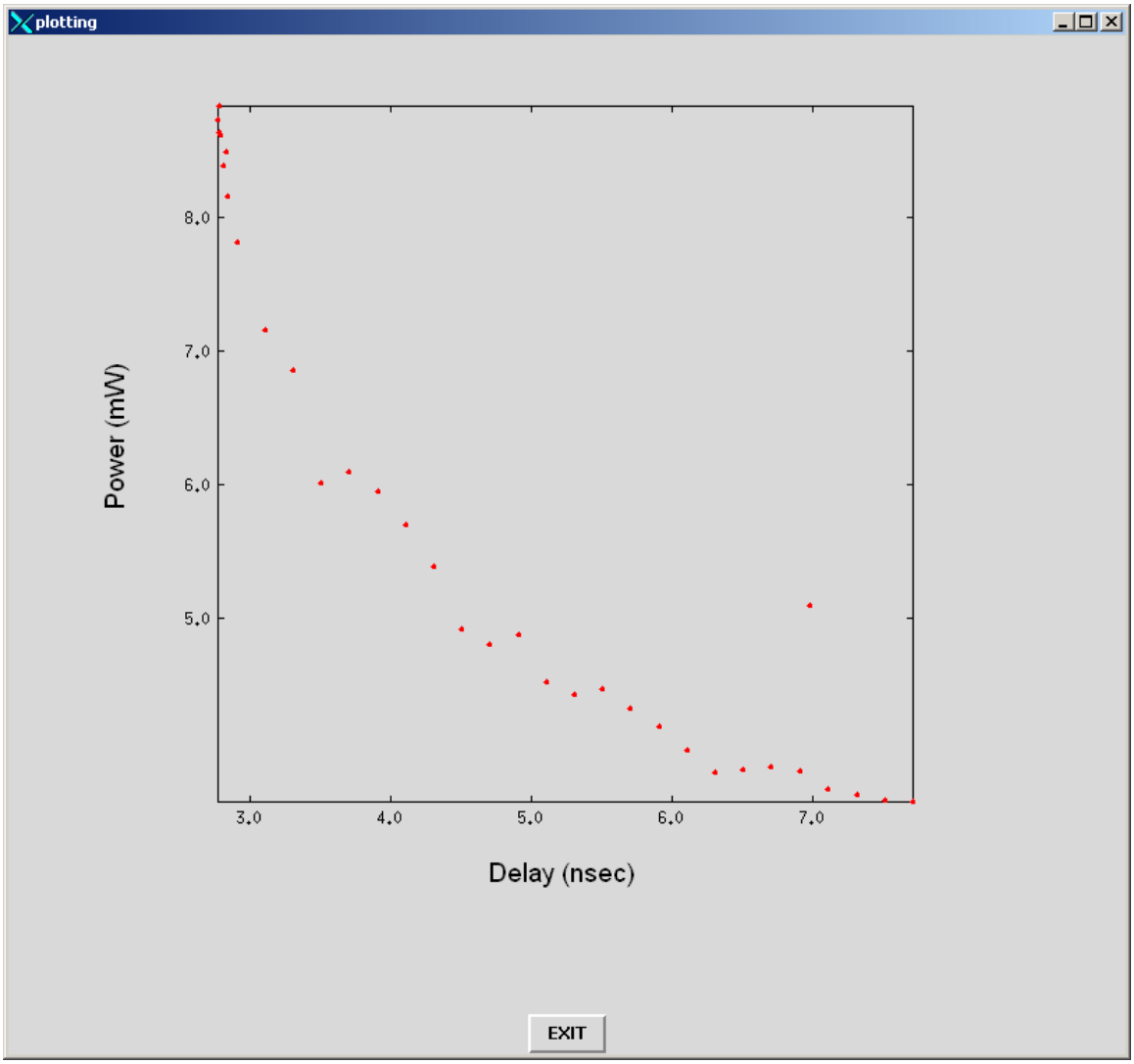

Figure 4.8: Power vs. Delay for 32-bit Multiplier

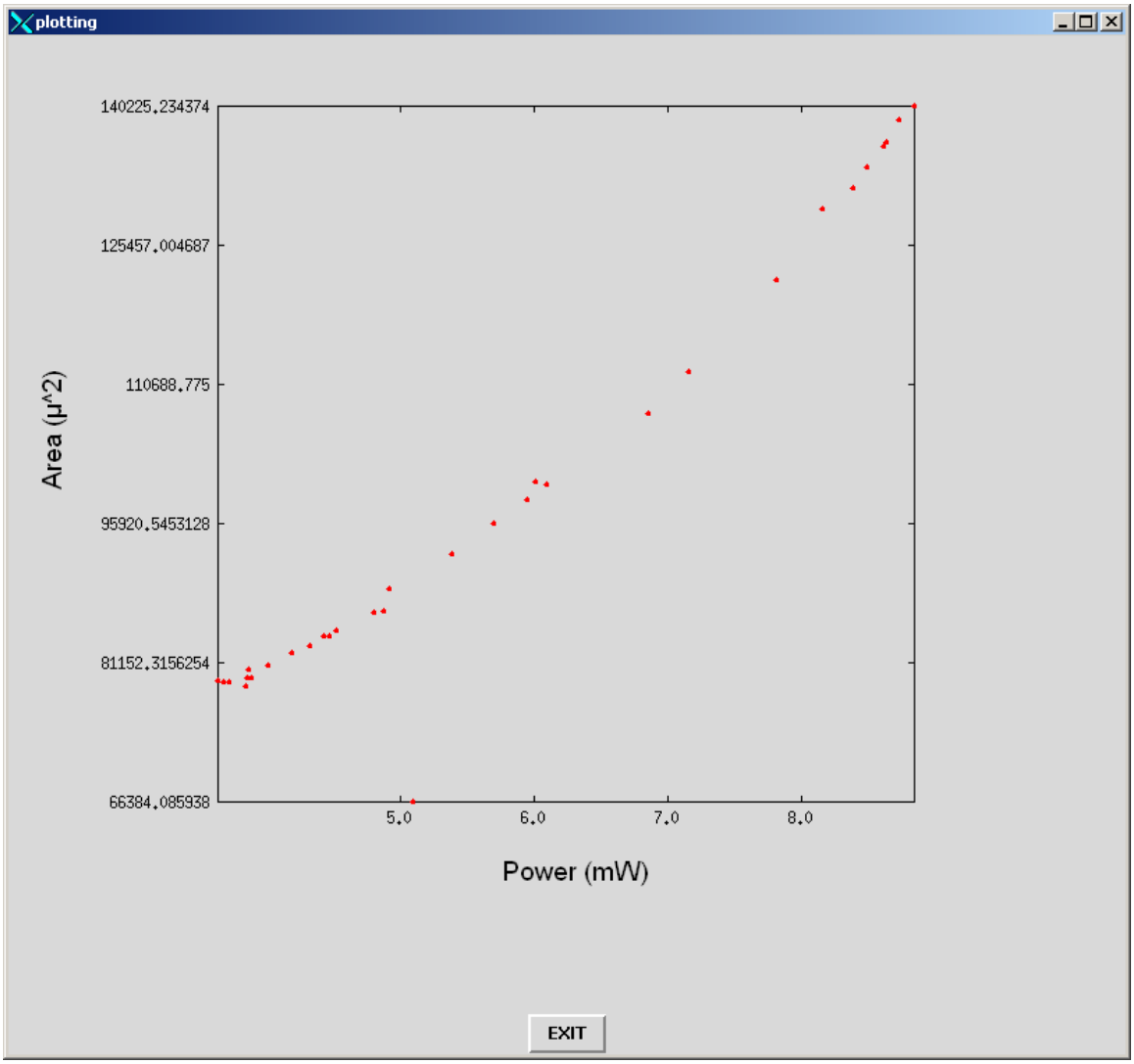

Figure 4.9: Area vs. Power for 32-bit Multiplier

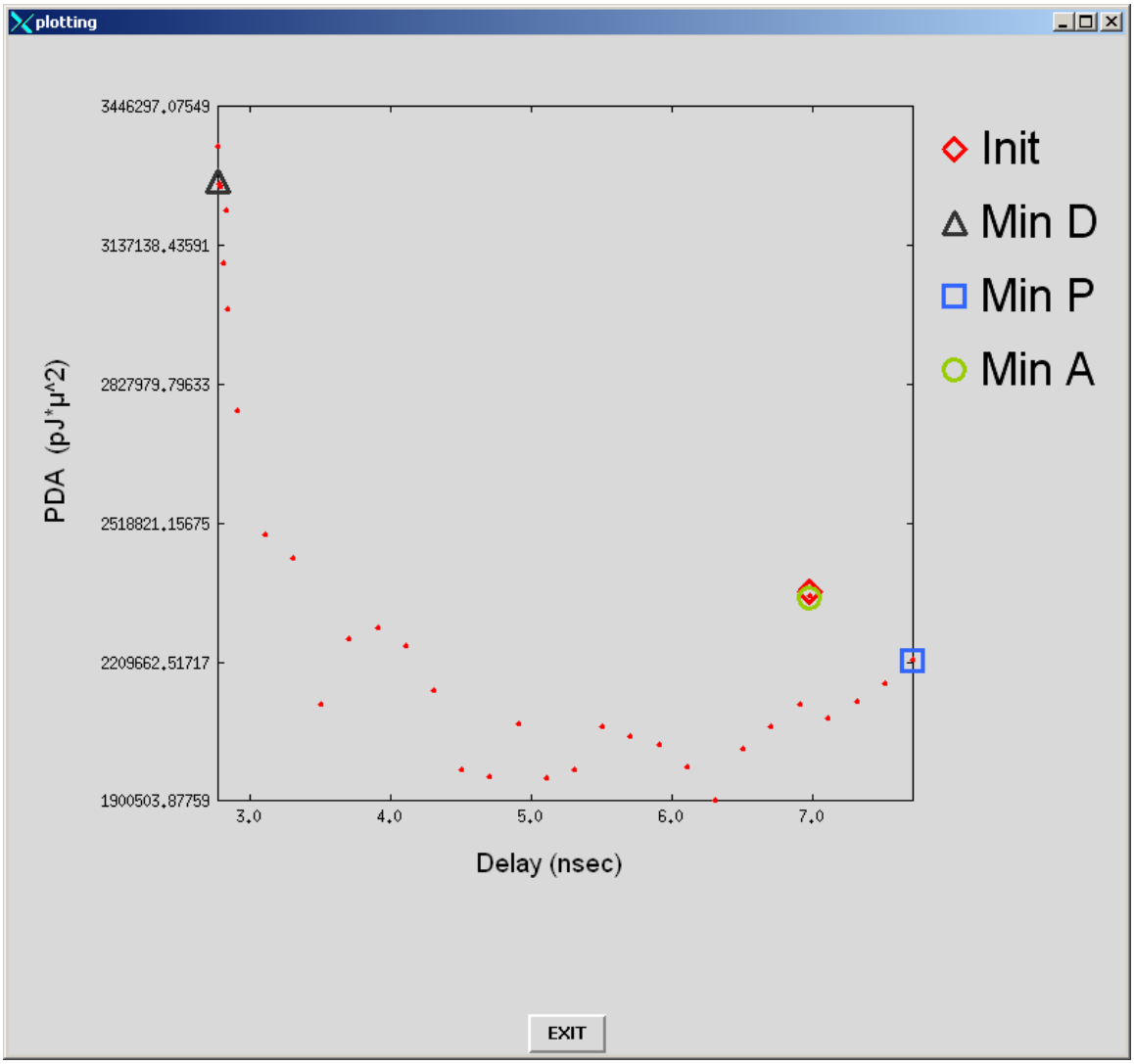

Figure 4.10: PDA vs. Delay for 32-bit Multiplier

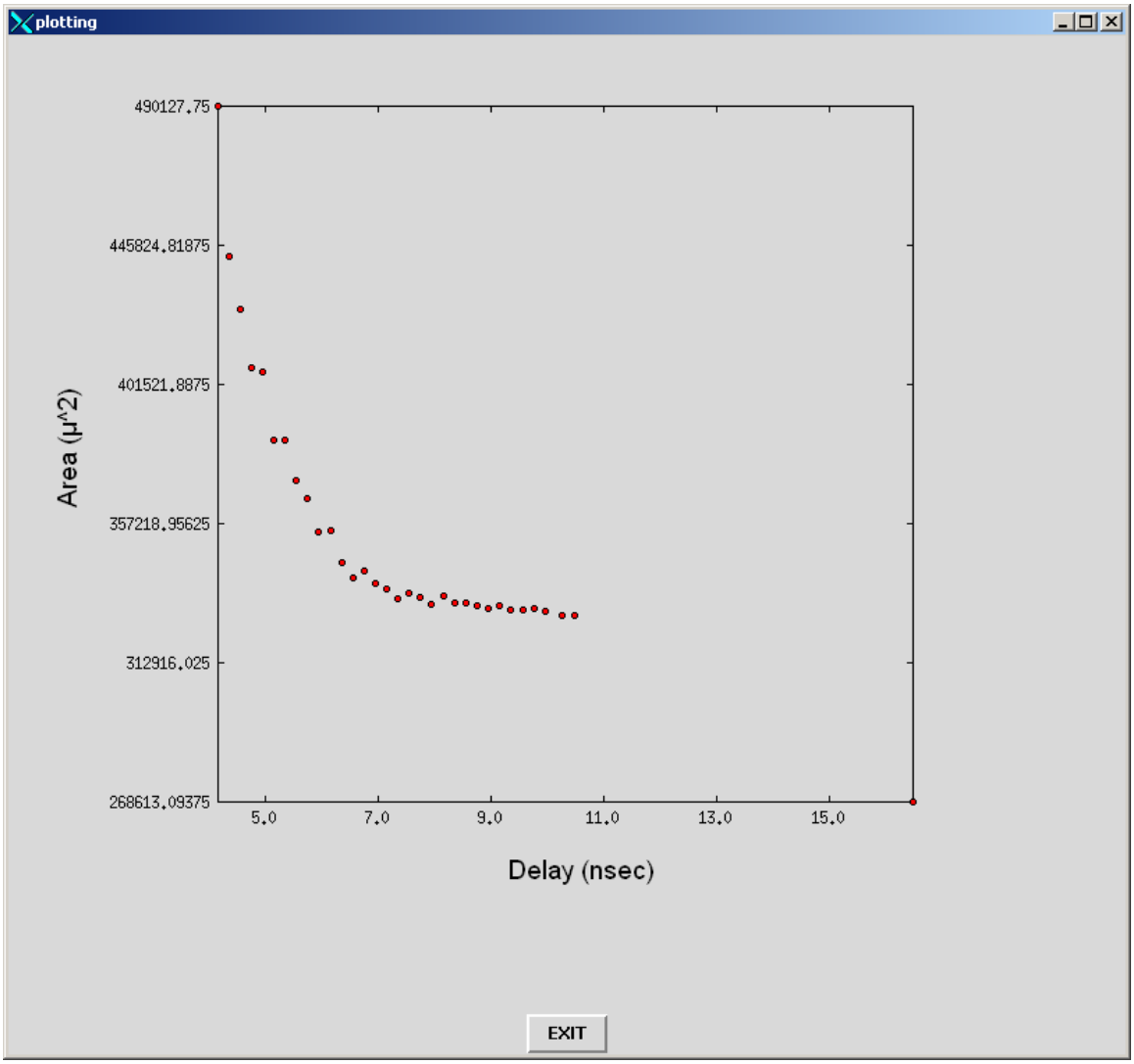

Figure 4.11: Area vs. Delay for 32-bit Complex Multiplier

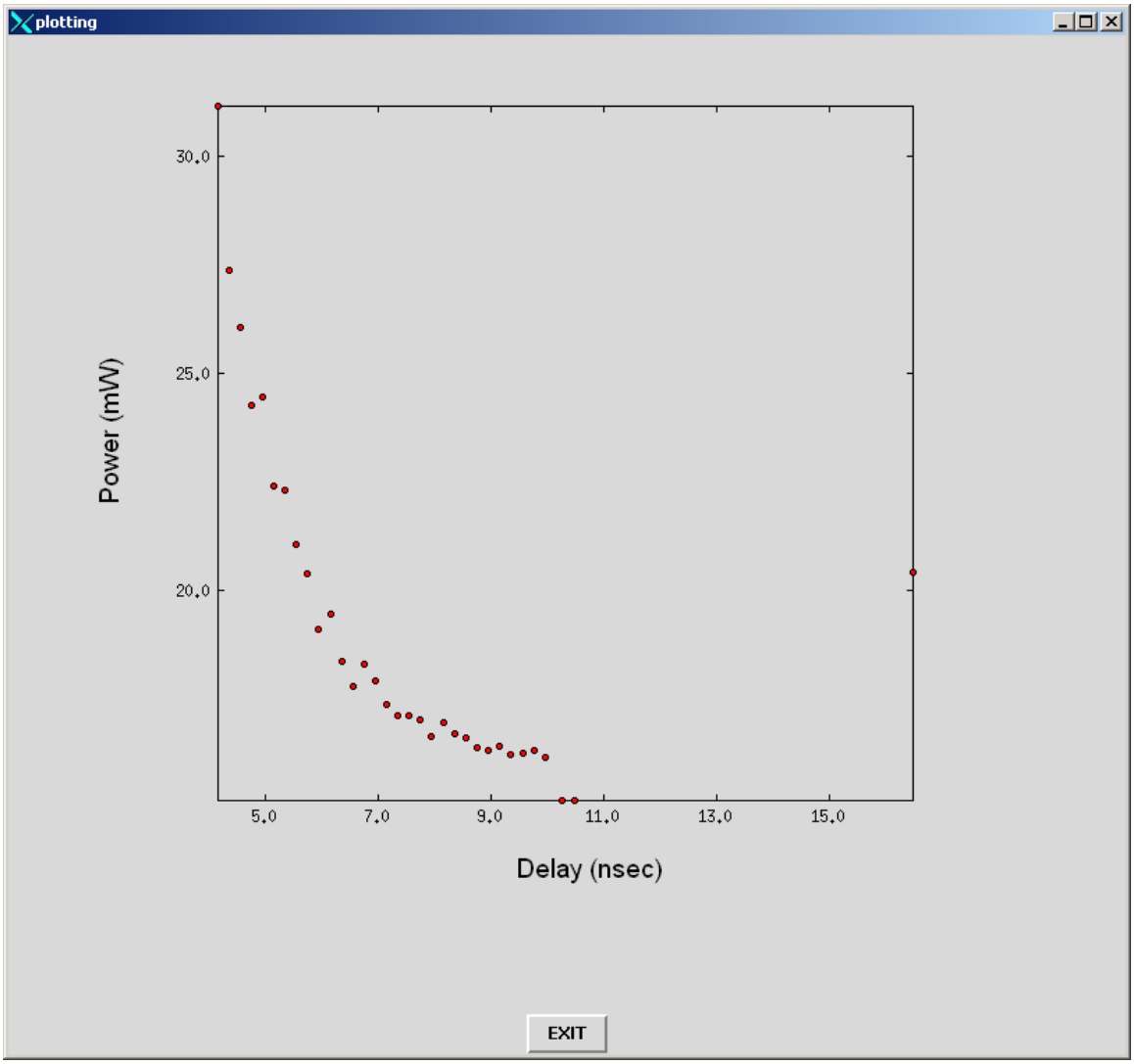

Figure 4.12: Power vs. Delay for 32-bit Complex Multiplier

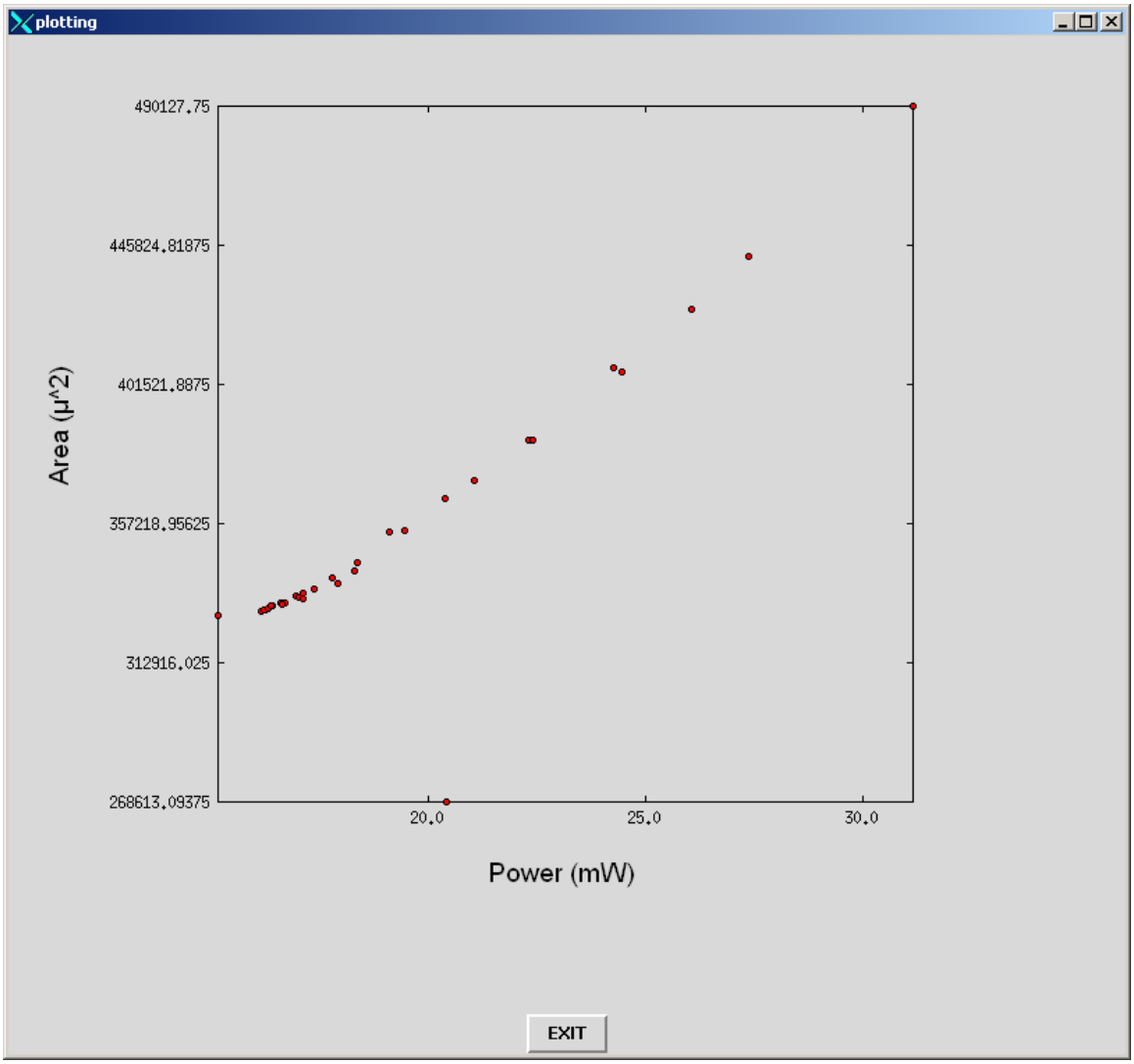

Figure 4.13: Area vs. Power for 32-bit Complex Multiplier

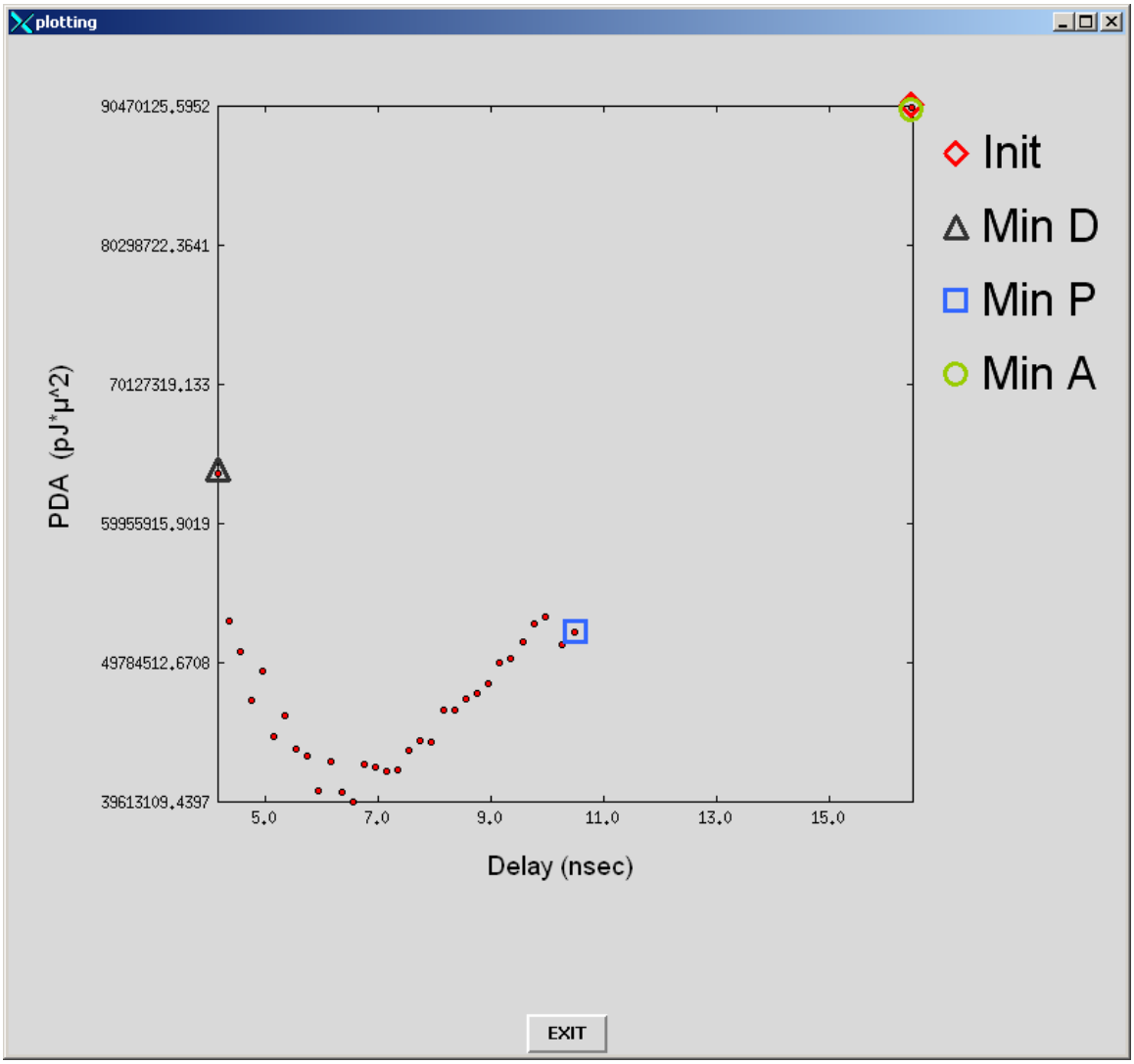

Figure 4.14: PDA vs. Delay for 32-bit Complex Multiplier

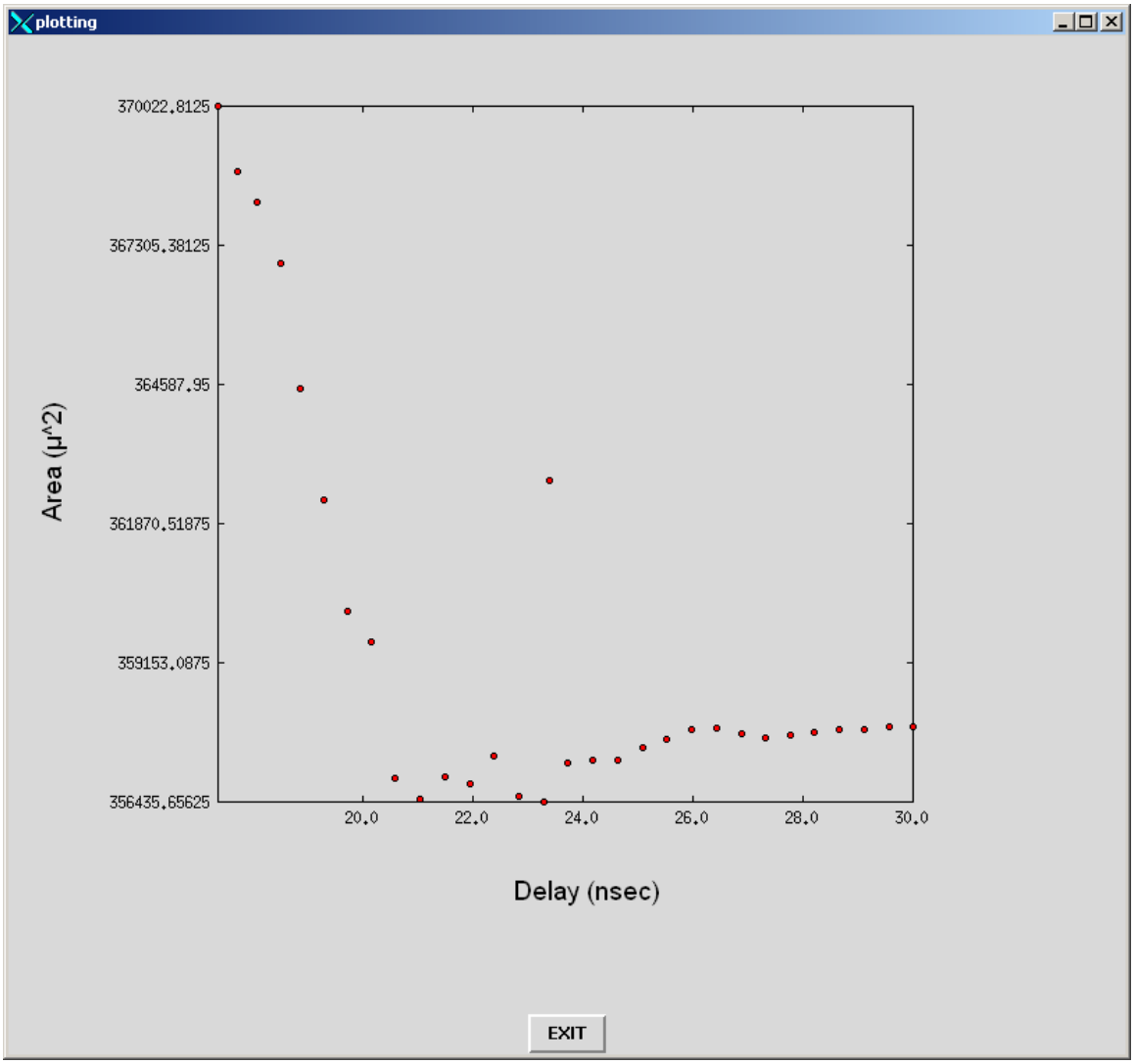

Figure 4.15: Area vs. Delay for 64-bit FFT

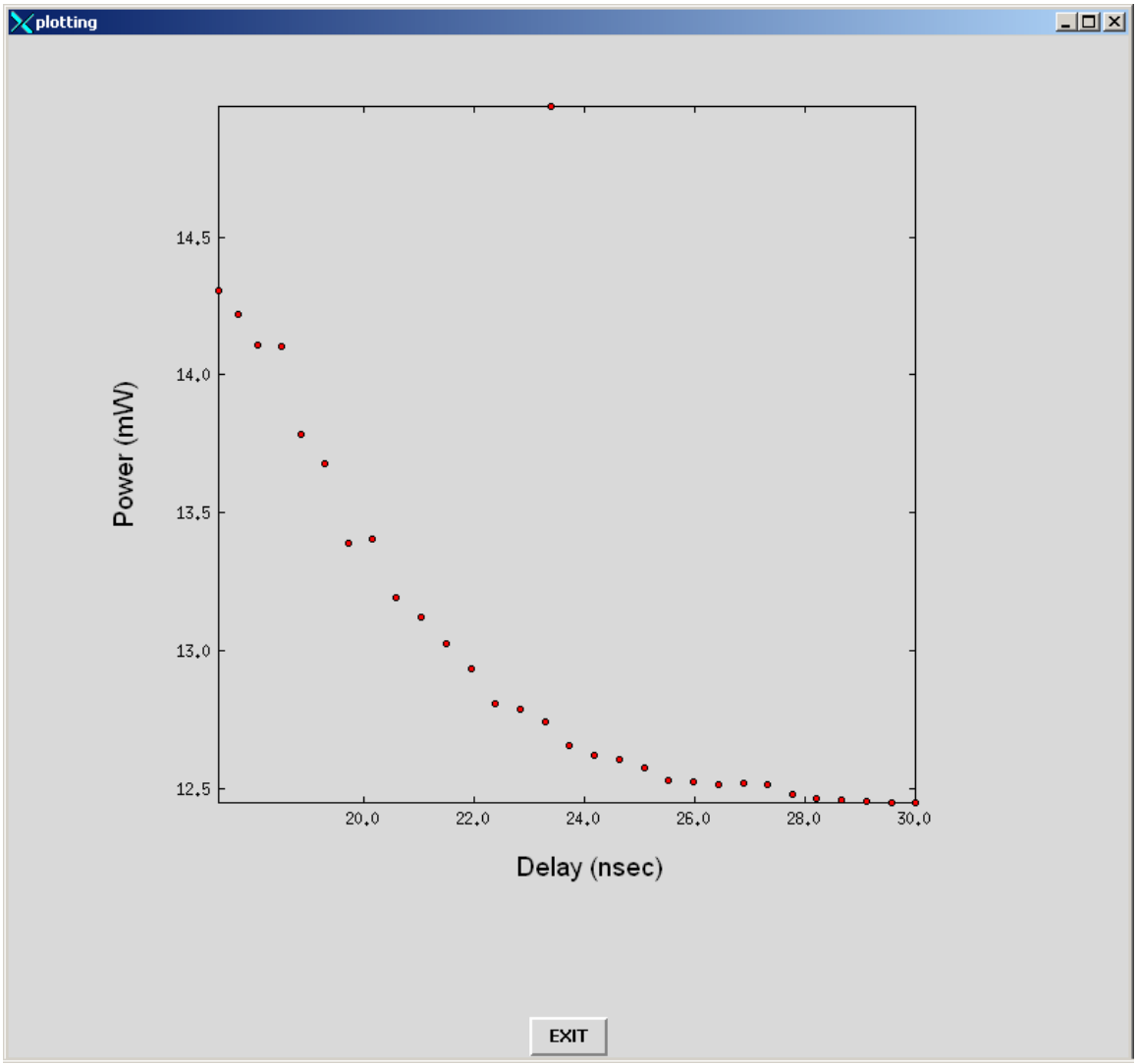

Figure 4.16: Power vs. Delay for 64-bit FFT

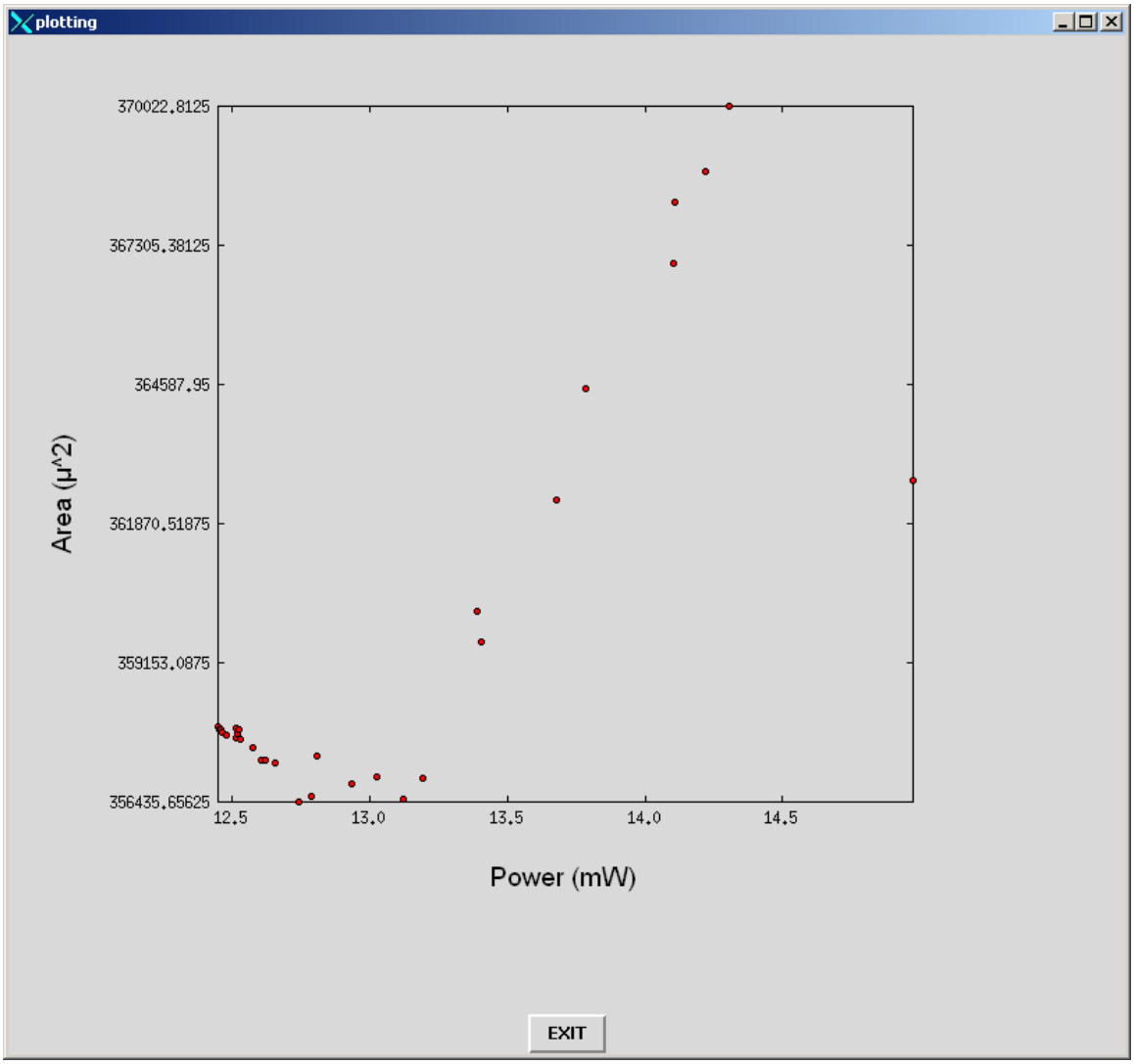

Figure 4.17: Area vs. Power for 64-bit FFT

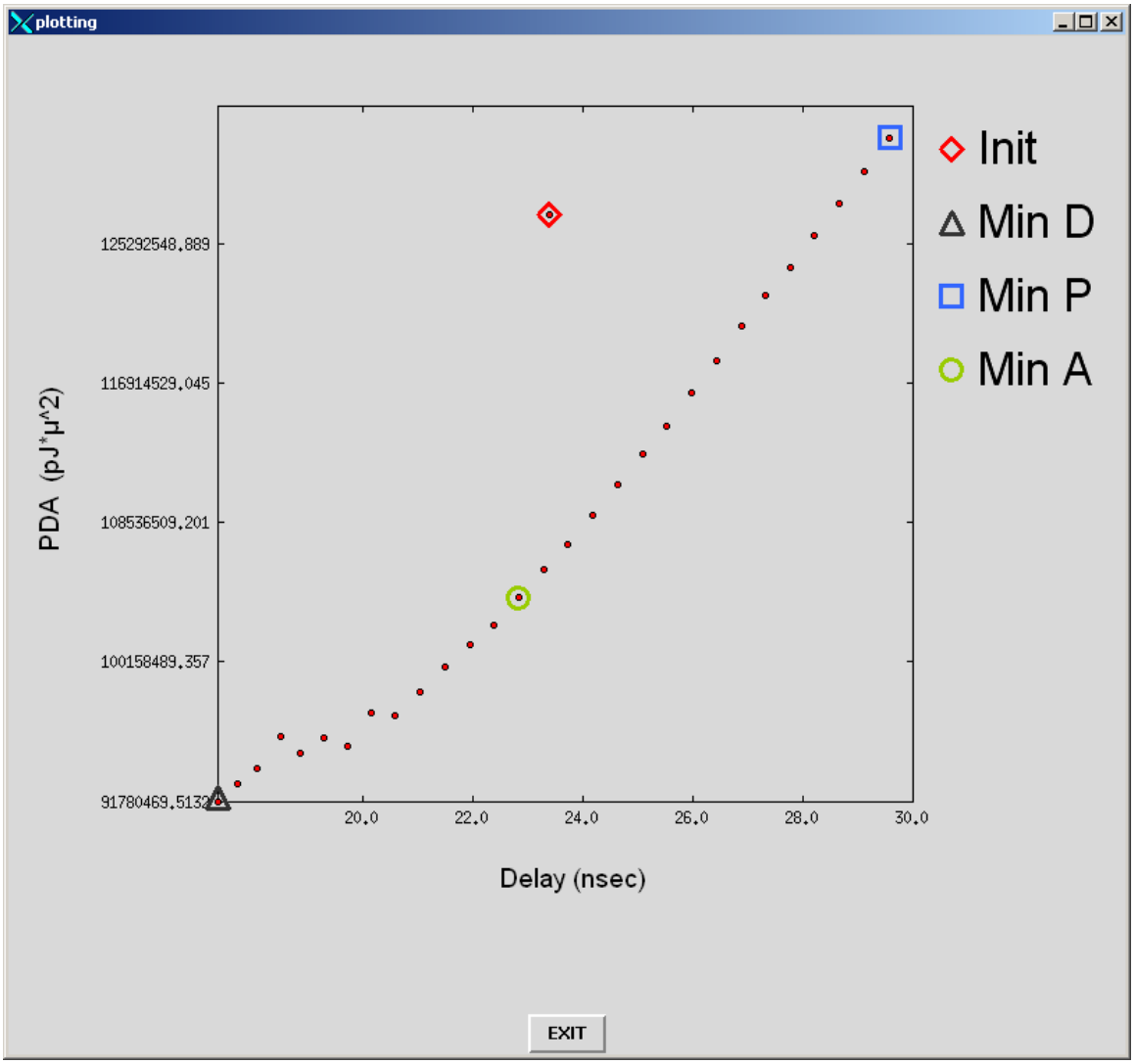

Figure 4.18: PDA vs. Delay for 64-bit FFT

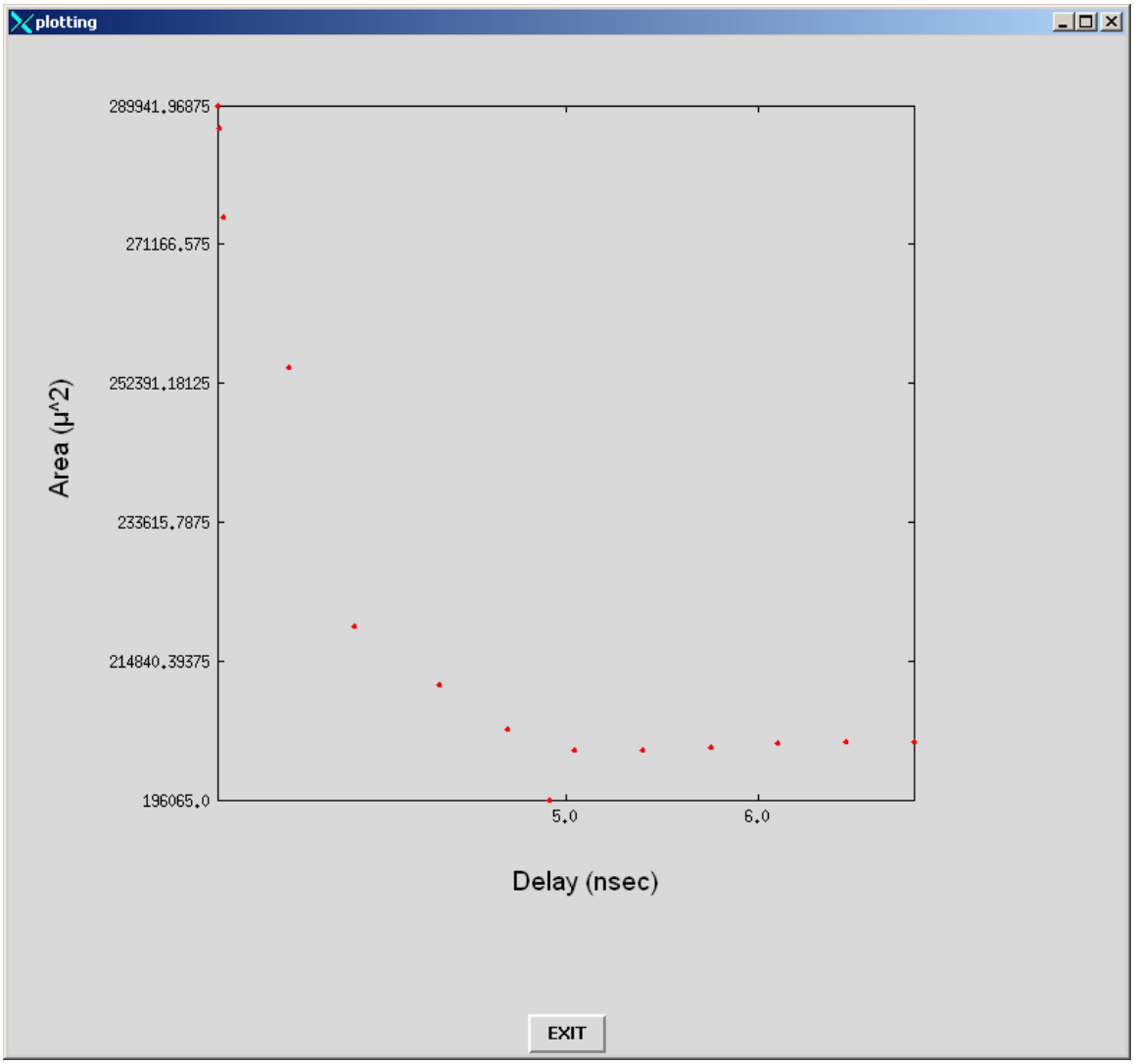

Figure 4.19: Area vs. Delay for 32-bit FIR

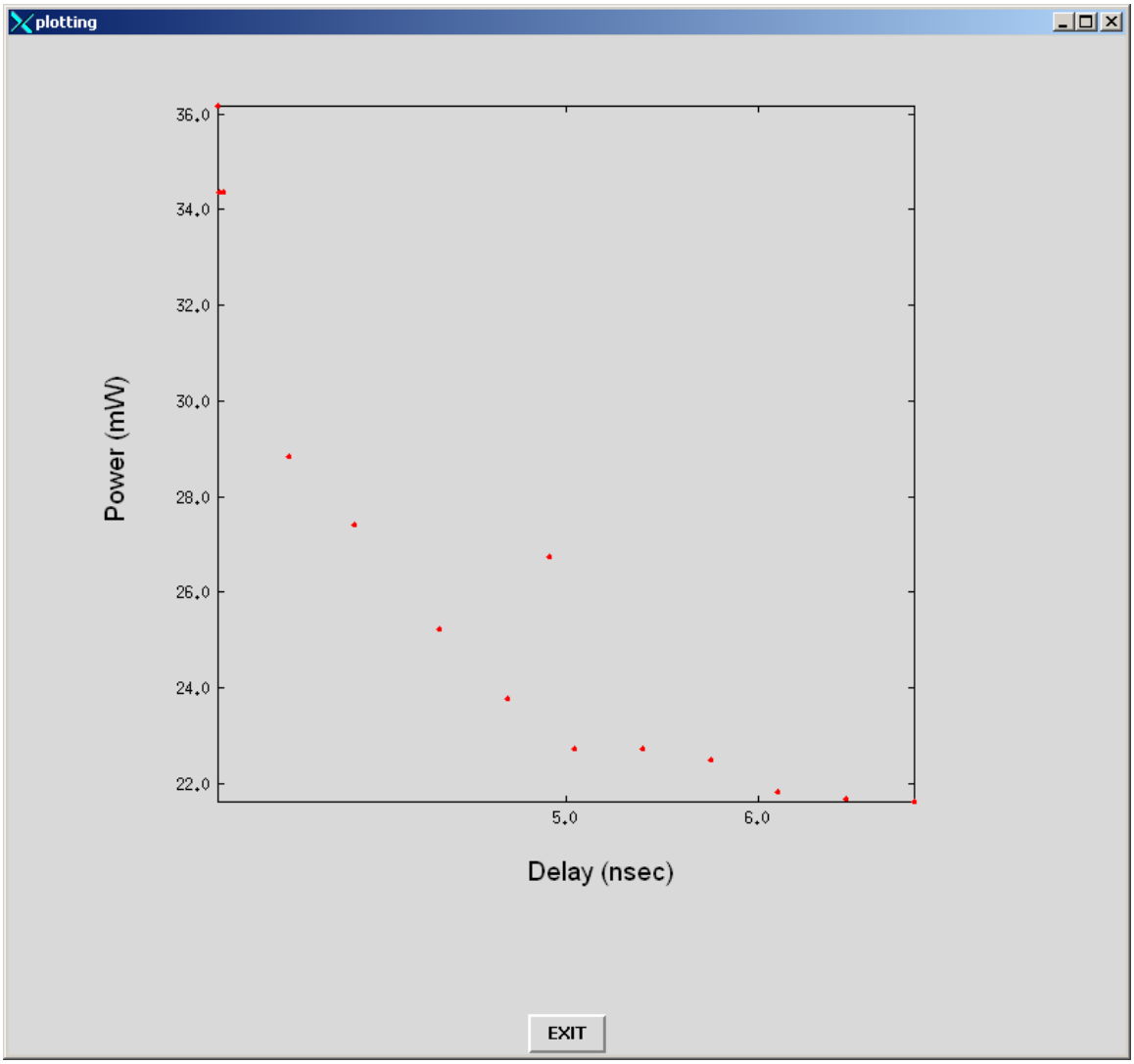

Figure 4.20: Power vs. Delay for 32-bit FIR

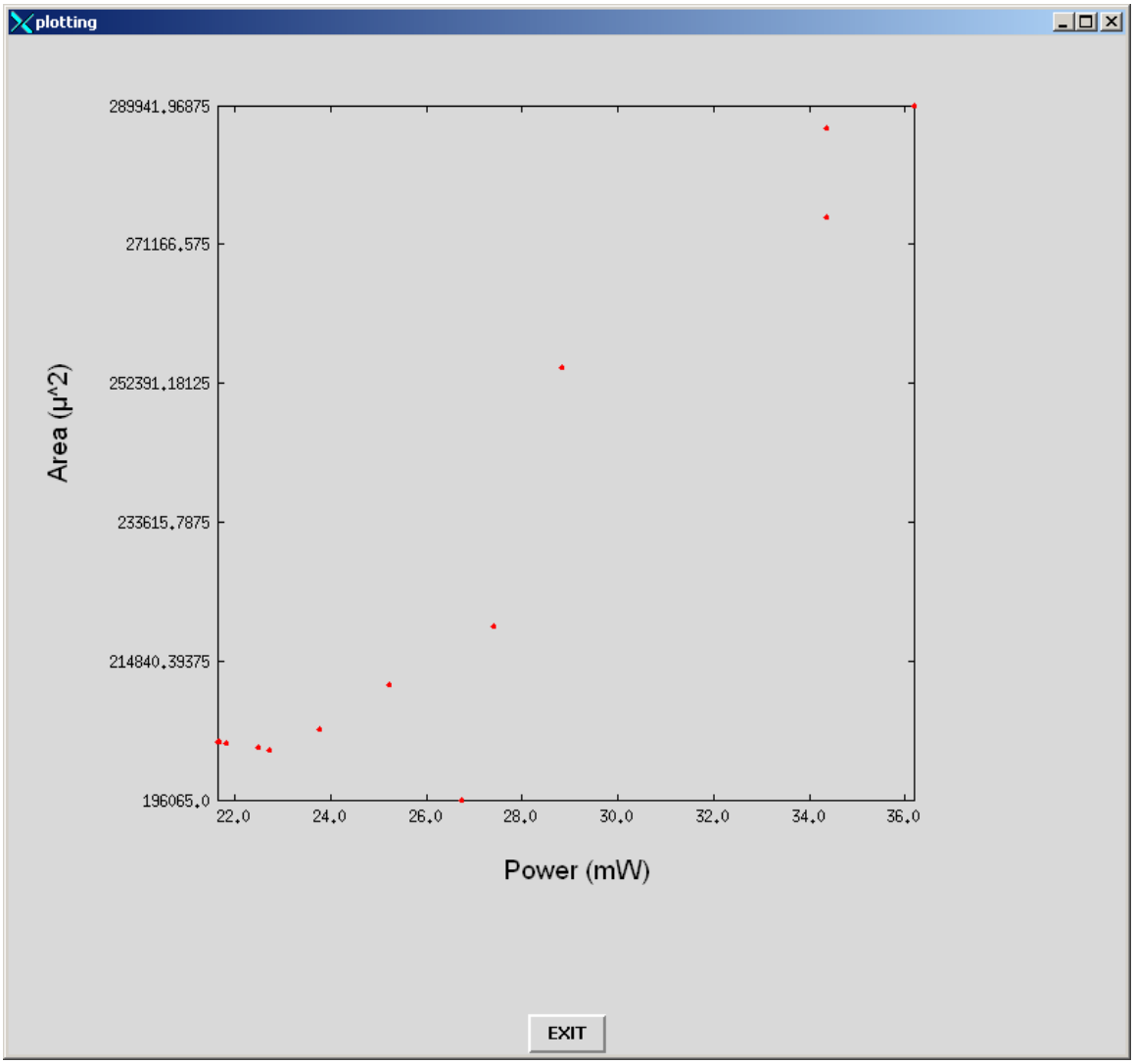

Figure 4.21: Area vs. Power for 32-bit FIR

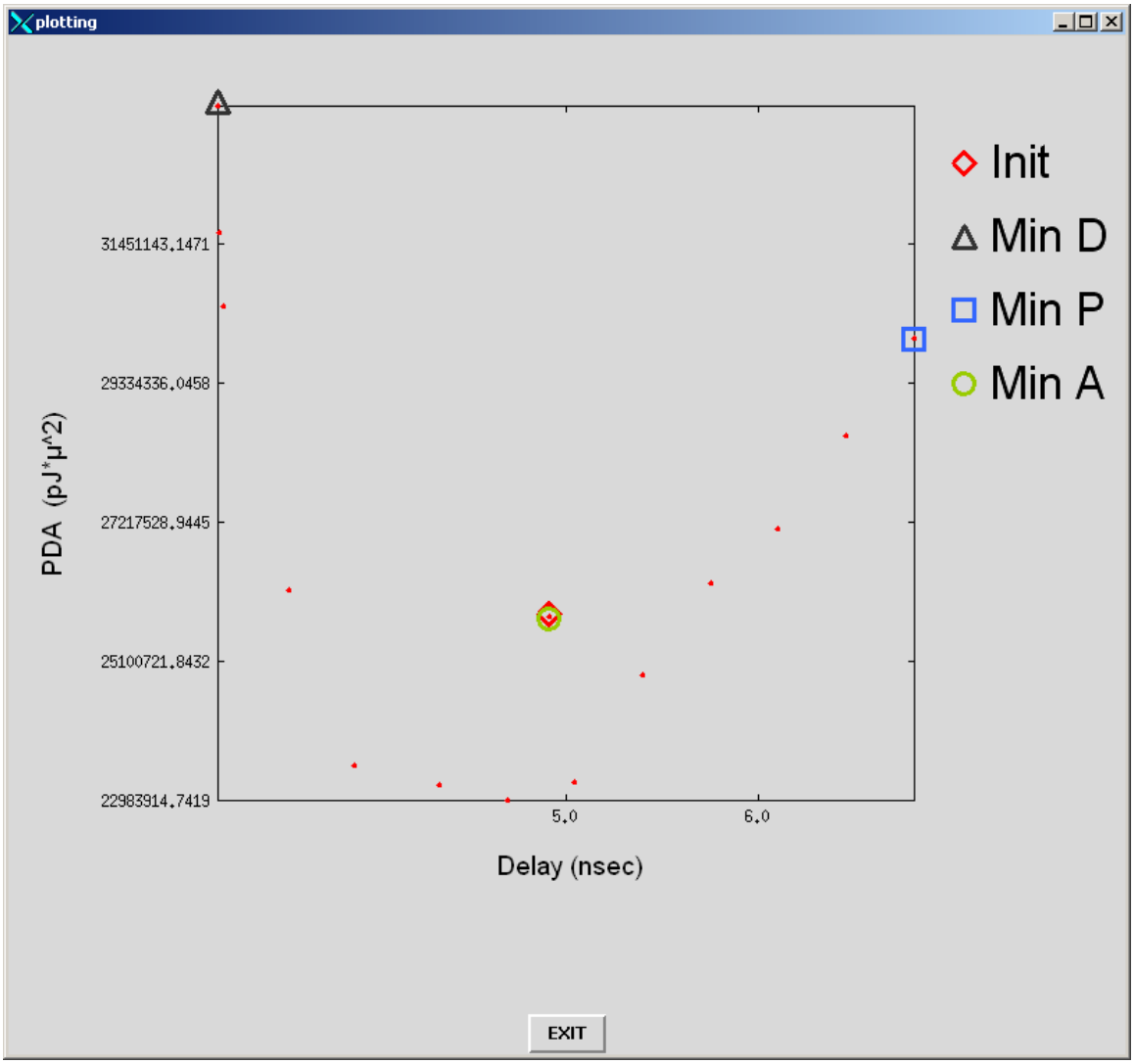

Figure 4.22: PDA vs. Delay for 32-bit FIR

Chapter 5 Discussion and Conclusions

Design optimization is an important step in implementation of any design. The ability to target specific areas of need can be a significant advantage for the designer. Having an automated process to perform the optimizations can reduce time to market and overhead costs associated with hand optimizations.

5.1 Synplicity

Based on the results from numerous experiments, Synplicity's Synplify and Amplify suites do not appear to give adequate results for automated optimization. The tools are developed with the design engineer in mind, giving him full control over synthesis, placement and routing optimizations. Without this guiding hand, the tools attempt to create the best solution possible in the least amount of time. The tools will even try solutions outside of the initialization parameters in order to present the best solution even with poor initialization values. This further shows that the tool was developed with the design engineer in mind.

5.2 Liberator

Liberator, on the other hand, is well suited for optimizing ASIC designs within its circuit list. The DSSA attempts to find the global minima for power, delay and area, and the Liberator tool implements this nicely.

Chapter 6 Future Work

This paper has detailed how to use the Synplicity physical synthesis toolsets for FPGAs on the UT Solaris workstation cluster, evaluation of the Synplicity physical synthesis toolsets to determine their efficacy in achieving multiple solutions over the power-delay-area design space and evaluation of the Liberator toolset for ASIC design to determine its efficacy in achieving multiple solutions over the power-delay-area design space.

Synplicity's Synplify and Amplify have proven to be valuable optimization tools for the design engineer. However, their use as an automated optimization tool to provide multiple solutions targeting differing needs is lacking. To fully utilize the tool, the design engineer must interact with and guide the software to obtain the desired results.

The Liberator software is extremely useful for generating multiple solutions targeting power, delay and area, or combinations thereof. However, the Liberator software currently works only with its predefined user-configurable circuits. The underlying interface for Liberator needs modification to allow importation of external designs.

List of References

List of References

[1] Karakaya, F., "Automated Exploration of the ASIC Design Space for Minimum Power-Delay- Area Product at the Register Transfer Level," Ph.D. Dissertation, University of Tennessee, May 2004.

[2] Fields, S., "Hardware Design and Implementation of Role-Based Cryptography," M.S. Thesis, University of Tennessee, Dec. 2005.

[3] Snapp, W., Haug, P., Sunderland, D., Brees, R., Bouldin, D., Sechen, C. and W. Dai, "MSP Liberator ASIC Design Flow Produces Full Custom Performance Required for Next Generation Military Electronics," Proceedings of 2003 Government Microcircuit Applications Conference (GOMAC), pp. 498-501, Tampa, FL, April 3, 2003.

[4] Synplicity Synplify User's Guide. Synplicity. February 2004.

[5] Synplicity Synplify Pro User's Guide. Synplicity. February 2004.

[6] Synplicity Amplify FPGA Physical Optimizer User's Guide. Synplicity. February 2004.

[7] Balakrishnan, A., "An Experimental Study of the Accuracy of Multiple Power Estimation Methods," M.S. Thesis, University of Tennessee, August 2004

Vita

Jonathan Turnmire was born in Knoxville, TN on February 2, 1982. He graduated from South Doyle High School in 1999. He then went to the University of Tennessee, Knoxville where he studied Computer Engineering. He earned a Bachelor of Science degree Summa cum Laude in 2003. Jonathan remained at the University of Tennessee in Knoxville for his graduate studies. He received a Master of Science degree for Electrical Engineering in 2006 and is currently pursuing his doctorate in Computer Engineering.